

Formulations and solutions of IMO problems in Isabelle/HOL

Filip Marić and Sana Stojanović-Durđević

June 11, 2020

Contents

1	IMO 2006 SL statements	5
1.1	Algebra problems	5
1.1.1	IMO 2006 SL - A1	5
1.1.2	IMO 2006 SL - A2	6
1.1.3	IMO 2006 SL - A3	6
1.1.4	IMO 2006 SL - A4	6
1.1.5	IMO 2006 SL - A5	7
1.1.6	IMO 2006 SL - A6	7
2	IMO 2008 SL statements	9
2.1	Algebra problems	9
2.1.1	IMO 2008 SL - A1	9
3	IMO 2018 SL statements	11
3.1	Algebra problems	11
3.1.1	IMO 2018 SL - A1	11
3.1.2	IMO 2018 SL - A2	12
3.1.3	IMO 2018 SL - A3	12
3.1.4	IMO 2018 SL - A4	12
3.1.5	IMO 2018 SL - A5	13
3.1.6	IMO 2018 SL - A7	13
4	IMO 2006 SL solutions	15
4.1	Algebra problems	15
4.1.1	IMO 2006 SL - A2	15
5	IMO 2018 SL solutions	19
5.1	Algebra problems	19

5.1.1	IMO 2018 SL - A2	19
5.1.2	IMO 2018 SL - A4	29
5.2	Combinatorics problems	47
5.2.1	IMO 2018 SL - C1	47
5.2.2	IMO 2018 SL - C2	59
5.2.3	IMO 2018 SL - C3	83
5.2.4	IMO 2018 SL - C4	138
5.3	Number theory problems	166
5.3.1	IMO 2018 SL - N5	166

Chapter 1

IMO 2006 SL statements

```
theory IMO-2006-SL-statements  
  imports Main
```

```
begin
```

Shortlisted problems with solutions from *47-th International Mathematical Olympiad, 2006, Slovenia*.

File with problem statements and solutions can be found at: <https://www.imo-official.org/problems/IMO2006SL.pdf>

```
end
```

1.1 Algebra problems

1.1.1 IMO 2006 SL - A1

```
theory IMO-2006-SL-A1  
imports Complex-Main  
begin
```

```
theorem IMO-2006-SL-A1:
```

```
  fixes  $a :: nat \Rightarrow real$ 
```

```
  assumes  $\forall i \geq 0. (a\ i + 1) = floor\ (a\ i) * (a\ i - floor\ (a\ i))$ 
```

```
  shows  $\exists j. a\ j = a\ (j + 2)$ 
```

```
  sorry
```

```
end
```

1.1.2 IMO 2006 SL - A2

theory *IMO-2006-SL-A2*

imports *Complex-Main*

begin

theorem *IMO-2006-SL-A2*:

fixes $a :: \text{nat} \Rightarrow \text{real}$

assumes $a\ 0 = -1 \ \forall \ n \geq 1. (\sum \ k < \text{Suc } n. a\ (n - k) / (k + 1)) = 0 \ n \geq 1$

shows $a\ n > 0$

using $\langle n \geq 1 \rangle$

sorry

end

1.1.3 IMO 2006 SL - A3

theory *IMO-2006-SL-A3*

imports *Complex-Main Tutorial.Pairs*

begin

theorem *IMO-2006-SL-A3*:

fixes $c :: \text{nat} \Rightarrow \text{nat}$

and $S :: (\text{nat} \times \text{nat}) \text{ set}$

assumes $c\ 0 = 1 \ c\ 1 = 0 \ \forall \ n \geq 0. (c\ (n + 2) = c\ (n + 1) + c\ n)$ **and**

$\forall \ (x, y) \in S. \exists \ J :: \text{nat set}. (\forall \ j \in J. j > 0) \wedge$

$(\sum_{j \in J} c\ j) = x \wedge (\sum_{j \in J} c\ (j - 1)) = y$

shows $\exists \ \alpha \ \beta \ m \ M ::$

$\text{real}. (x, y) \in S \longleftrightarrow (m < \alpha * x + \beta * y \wedge \alpha * x + \beta * y < M)$

sorry

end

1.1.4 IMO 2006 SL - A4

theory *IMO-2006-SL-A4*

imports *Complex-Main*

begin

theorem *IMO-2006-SL-A4*:

```

fixes  $a :: nat \Rightarrow real$  and  $n :: nat$ 
assumes
   $\forall i. 1 \leq i \wedge i \leq n \wedge a\ i > 0$ 
   $\forall n \geq 1. (\sum k < Suc\ n. a\ (n - k) / (k + 1)) = 0\ n \geq 1$ 
shows
   $(\sum i \leftarrow [1..<n]. \sum j \leftarrow [i+1..<n+1]. (a\ i * a\ j / (a\ i + a\ j))) \leq$ 
   $(n / (2 * (\sum i \leftarrow [1..<n+1]. a\ i)) * (\sum i \leftarrow [1..<n]. \sum j \leftarrow [i+1..<n+1].$ 
   $(a\ i * a\ j)))$ 
sorry

end

```

1.1.5 IMO 2006 SL - A5

```

theory IMO-2006-SL-A5
imports Complex-Main HOL-ex.Sqrt
begin

theorem IMO-2006-SL-A5:
  fixes  $a\ b\ c :: nat$ 
  assumes  $a > 0\ b > 0\ c > 0\ a + b > c\ b + c > a\ c + a > b$ 
  shows  $\sqrt{b + c - a} / (\sqrt{b} + \sqrt{c} - \sqrt{a}) +$ 
   $\sqrt{c + a - b} / (\sqrt{c} + \sqrt{a} - \sqrt{b}) +$ 
   $\sqrt{a + b - c} / (\sqrt{a} + \sqrt{b} - \sqrt{c}) \leq 3$ 
sorry

end

```

1.1.6 IMO 2006 SL - A6

```

theory IMO-2006-SL-A6
imports Complex-Main HOL-ex.Sqrt
begin

theorem IMO-2006-SL-A6:
  fixes  $a\ b\ c :: real$ 
  shows  $Max\ \{(a * b * (a * a - b * b) + b * c * (b * b - c * c) + c * a * (c * c - a * a)) / (a * a + b * b + c * c)\} = (\sqrt{2}) * 9 / 32$ 
sorry

```

end

Chapter 2

IMO 2008 SL statements

```
theory IMO-2008-SL-statements  
  imports Main
```

```
begin
```

Shortlisted problems with solutions from *49-th International Mathematical Olympiad, July 10-22, 2008, Madrid, Spain.*

File with problem statements and solutions can be found at: <https://www.imo-official.org/problems/IMO2008SL.pdf>

```
end
```

2.1 Algebra problems

2.1.1 IMO 2008 SL - A1

```
theory IMO-2008-SL-A1  
imports Complex-Main  
begin
```

```
theorem IMO-2008-SL-A1:
```

```
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
```

```
  assumes  $\forall p\ q\ r\ s :: \text{real}. p > 0 \wedge q > 0 \wedge r > 0 \wedge s > 0 \wedge pq = rs \wedge$   
     $((f\ p)^2 + (f\ q)^2) / ((f\ r)^2 + (f\ s)^2) = (p^2 + q^2) / (r^2 + s^2)$ 
```

```
  shows  $(\forall x > 0. f\ x = x) \vee (\forall x > 0. f\ x = 1 / x)$ 
```

```
  sorry
```

end

Chapter 3

IMO 2018 SL statements

```
theory IMO-2018-SL-statements  
  imports Main
```

```
begin
```

Shortlisted problems with solutions from *59-th International Mathematical Olympiad, 3-14 July 2018, Cluj-Napoca, Romania*.

File with problem statements and solutions can be found at: <https://www.imo-official.org/problems/IMO2018SL.pdf>

```
end
```

3.1 Algebra problems

3.1.1 IMO 2018 SL - A1

```
theory IMO-2018-SL-A1  
imports HOL.Rat
```

```
begin
```

```
theorem IMO2018SL-A1:
```

```
  fixes  $x\ y :: \text{rat}$  and  $f :: \text{rat} \Rightarrow \text{rat}$ 
```

```
  assumes  $f\ (x * x * (f\ y) * (f\ y)) = (f\ x) * (f\ x) * (f\ y)$ 
```

```
  shows  $f\ x = 1$ 
```

```
  sorry
```

end

3.1.2 IMO 2018 SL - A2

theory *IMO-2018-SL-A2*
 imports *Complex-Main*
 begin

theorem *IMO2018SL-A2*:

fixes $n :: \text{nat}$

assumes $n \geq 3$

shows $(\exists a :: \text{nat} \Rightarrow \text{real}. a\ n = a\ 0 \wedge a\ (n+1) = a\ 1 \wedge$
 $(\forall i < n. (a\ i) * (a\ (i+1)) + 1 = a\ (i+2))) \longleftrightarrow$
 $3\ \text{dvd}\ n\ (\text{is}\ (\exists a. ?p1\ a \wedge ?p2\ a \wedge ?eq\ a) \longleftrightarrow 3\ \text{dvd}\ n)$

sorry

end

3.1.3 IMO 2018 SL - A3

theory *IMO-2018-SL-A3*
 imports *Complex-Main*

begin

theorem *IMO2018SL-A3*:

fixes $S :: \text{nat set}$

assumes $\forall x \in S. x > 0$

shows $(\exists F\ G. F \subseteq S \wedge G \subseteq S \wedge F \cap G = \{\} \wedge (\sum_{x \in F}. 1/(\text{rat-of-nat}\ x))$
 $= (\sum_{x \in G}. 1/(\text{rat-of-nat}\ x))) \vee$
 $(\exists r :: \text{rat}. 0 < r \wedge r < 1 \wedge (\forall F \subseteq S. \text{finite}\ F \longrightarrow (\sum_{x \in F}. 1/(\text{rat-of-nat}\ x)) \neq r))$

sorry

end

3.1.4 IMO 2018 SL - A4

theory *IMO-2018-SL-A4*
 imports *Complex-Main*

begin

definition *is-Max* :: 'a::linorder set \Rightarrow 'a \Rightarrow bool **where**
is-Max A x \longleftrightarrow x \in A \wedge (\forall x' \in A. x' \leq x)

theorem *IMO2018SL-A4*:

shows

is-Max {a 2018 - a 2017 | a::nat \Rightarrow real. a 0 = 0 \wedge a 1 = 1 \wedge (\forall n \geq 2. \exists k. 1 \leq k \wedge k \leq n \wedge a n = (\sum i \leftarrow [n-k.. $<$ n]. a i) / real k)}
 (2016 / 2017²) (**is** *is-Max* {?f a | a. ?P a} ?m)

unfolding *is-Max-def*

sorry

end

3.1.5 IMO 2018 SL - A5

theory *IMO-2018-SL-A5*

imports *Complex-Main*

begin

theorem *IMO-2018-SL-A5*:

fixes f :: real \Rightarrow real

assumes \forall x $>$ 0. \forall y $>$ 0. (x + 1/x) * (f y) = f (x*y) + f (y / x)

shows \exists C1 C2. \forall x $>$ 0. f x = C1 * x + C2 / x

sorry

end

3.1.6 IMO 2018 SL - A7

theory *IMO-2018-SL-A7*

imports *Complex-Main*

begin

theorem

shows Max {root 3 (a / (b + 7)) + root 3 (b / (c + 7)) + root 3 (c / (d + 7)) + root 3 (d / (a + 7))
 | a b c d :: real . a \geq 0 \wedge b \geq 0 \wedge c \geq 0 \wedge d \geq 0 \wedge a + b + c + d
 = 100} = 8 / root 3 7

sorry

end

Chapter 4

IMO 2006 SL solutions

```
theory IMO-2006-SL-solutions  
  imports Main
```

```
begin
```

Shortlisted problems with solutions from *57-th International Mathematical Olympiad, Slovenia, 2006*.

File with problem statements and solutions can be found at: <https://www.imo-official.org/problems/IMO2006SL.pdf>

```
end
```

4.1 Algebra problems

4.1.1 IMO 2006 SL - A2

```
theory IMO-2006-SL-A2-sol  
imports Complex-Main  
begin
```

```
declare  $[[smt-timeout = 20]]$ 
```

```
lemma sum-remove-zero:
```

```
  fixes  $n :: nat$ 
```

```
  assumes  $n > 0$ 
```

```
  shows  $(\sum k < n. f\ k) = f\ 0 + (\sum k \in \{1..<n\}. f\ k)$ 
```

```
  using assms
```

by (simp add: atLeast1-lessThan-eq-remove0 sum.remove)

theorem *IMO-2006-SL-A2*:

fixes $a :: nat \Rightarrow real$

assumes $a\ 0 = -1 \ \forall \ n \geq 1. (\sum k < Suc\ n. a\ (n - k) / (k + 1)) = 0 \ n \geq 1$

shows $a\ n > 0$

using $\langle n \geq 1 \rangle$

proof (induct n rule: less-induct)

case (less n)

show ?case

proof (cases n = 1)

case True

then have $a\ 1 = 1/2$

using $assms(1)\ assms(2)[rule-format, of\ n]$

by simp

then show ?thesis

using $\langle n = 1 \rangle$

by simp

next

case False

then have $n > 1$

using $\langle n \geq 1 \rangle$

by simp

then have $n - 1 \geq 1$

by simp

have $(\sum k < n. a\ k / (n - k)) = 0$ (**is** ?S1 = 0)

using $assms(2)[rule-format, of\ n - 1]\ \langle n > 1 \rangle\ \langle n - 1 \geq 1 \rangle$

using $sum.nat-diff-reindex[of\ \lambda\ k. a\ k / (n - k)\ n]$

by simp

moreover

have $(\sum k < Suc\ n. a\ k / (n + 1 - k)) = 0$ (**is** ?S2 = 0)

using $assms(2)[rule-format, of\ n]\ \langle n > 1 \rangle$

using $sum.nat-diff-reindex[of\ \lambda\ k. a\ k / (n + 1 - k)\ Suc\ n]$

by auto

ultimately

have $(n + 1)*?S2 - n*?S1 = 0$

by simp


```

then have  $(n + 1) * (a \ n + (\sum k < n. a \ k / (n + 1 - k))) = n * ?S1$ 
  by (simp add: add.commute)
then have  $(n + 1) * a \ n = n * (\sum k < n. a \ k / (n - k)) - (n + 1) * (\sum$ 
 $k < n. a \ k / (n + 1 - k))$ 
  by (simp add: algebra-simps)
also have  $\dots = (\sum k < n. n * a \ k / (n - k)) - (\sum k < n. (n + 1) * a \ k /$ 
 $(n + 1 - k))$ 
  apply (subst sum-distrib-left)
  apply (subst sum-distrib-left)
  by simp
also have  $\dots = (\sum k < n. n * a \ k / (n - k) - (n + 1) * a \ k / (n + 1 - k))$ 
  apply (subst sum-subtractf)
  by simp
also have  $\dots = (\sum k < n. a \ k * (n / (n - k) - (n + 1) / (n + 1 - k)))$ 
  by (simp add: algebra-simps)
also have  $\dots = (\sum k \in \{1..<n\}. a \ k * (n / (n - k) - (n + 1) / (n + 1 -$ 
 $k)))$ 
  using  $\langle n > 1 \rangle$ 
  apply (subst sum-remove-zero[of n])
  by auto
also have  $\dots > 0$ 
proof (rule sum-pos)
  show finite  $\{1..<n\}$ 
    by simp
next
  show  $\{1..<n\} \neq \{\}$ 
    using  $\langle n > 1 \rangle$ 
    by simp
next
  fix  $i$ 
  assume  $i \in \{1..<n\}$ 
  then have  $1 \leq i \wedge i < n$ 
    by auto
  then have  $(n / (n - i) - (n + 1) / (n + 1 - i)) > 0$ 
  proof–
    let  $?n = \text{real } n$  and  $?i = \text{real } i$ 
    have  $(?n / (?n - ?i) - (?n + 1) / (?n + 1 - ?i)) > 0$ 
    proof–
      have  $?n / (?n - ?i) - (?n + 1) / (?n + 1 - ?i) = ?i / ((?n - ?i) *$ 
 $(?n + 1 - ?i))$ 

```

```

      using *
      by (simp add: field-simps)
    then show ?thesis
      using *
      by simp
  qed
  then show ?thesis
    using *
    by (simp add: add.commute of-nat-diff)
  qed
  moreover
  have  $a\ i > 0$ 
    using  $\text{less}(1)[\text{of } i]\ \langle 1 \leq i \rangle\ \langle i < n \rangle$ 
    by simp
  ultimately
  show  $a\ i * (n / (n - i) - (n + 1) / (n + 1 - i)) > 0$ 
    by simp
  qed
  ultimately
  have  $(n + 1) * a\ n > 0$ 
    by simp
  then show ?thesis
    by (smt mult-nonneg-nonpos of-nat-0-le-iff)
  qed
qed

end

```

Chapter 5

IMO 2018 SL solutions

```
theory IMO-2018-SL-solutions  
  imports Main
```

```
begin
```

Shortlisted problems with solutions from *59-th International Mathematical Olympiad, 3-14 July 2018, Cluj-Napoca, Romania*.

File with problem statements and solutions can be found at: <https://www.imo-official.org/problems/IMO2018SL.pdf>

```
end
```

5.1 Algebra problems

5.1.1 IMO 2018 SL - A2

```
theory IMO-2018-SL-A2-sol  
imports Complex-Main  
begin
```

```
lemma n-plus-1-mod-n:  
  fixes  $n :: \text{nat}$   
  assumes  $n > 1$   
  shows  $(n + 1) \bmod n = 1$   
  by (metis assms mod-add-self1 mod-less)
```

```
lemma n-plus-2-mod-n:
```

```

fixes  $n :: \text{nat}$ 
assumes  $n > 2$ 
shows  $(n + 2) \bmod n = 2$ 
by (metis assms mod-add-self1 mod-less)

```

theorem *IMO2018SL-A2*:

```

fixes  $n :: \text{nat}$ 
assumes  $n \geq 3$ 
shows  $(\exists a :: \text{nat} \Rightarrow \text{real}. a \bmod n = a \bmod 3 \wedge a \bmod (n+1) = a \bmod 2 \wedge$ 
 $(\forall i < n. (a \bmod i) * (a \bmod (i+1)) + 1 = a \bmod (i+2))) \longleftrightarrow$ 
 $3 \mid n \text{ (is } (\exists a. ?p1\ a \wedge ?p2\ a \wedge ?eq\ a) \longleftrightarrow 3 \mid n))$ 

```

proof

```

assume  $3 \mid n$ 

```

```

let  $?a = (\lambda n. \text{if } n \bmod 3 = 0 \text{ then } 2 \text{ else } -1) :: \text{nat} \Rightarrow \text{real}$ 
show  $\exists a. ?p1\ a \wedge ?p2\ a \wedge ?eq\ a$ 
proof (rule-tac  $x=?a$  in exI, safe)

```

```

  show  $?p1\ ?a$ 
  using  $\langle 3 \mid n \rangle$ 
  by auto

```

next

```

  show  $?p2\ ?a$ 
  using  $\langle 3 \mid n \rangle$ 
  by auto

```

next

```

  fix  $i$ 
  assume  $i < n$ 
  show  $(?a\ i) * (?a\ (i+1)) + 1 = ?a\ (i+2)$ 
  by auto presburger+

```

qed

next

```

assume  $\exists a. ?p1\ a \wedge ?p2\ a \wedge ?eq\ a$ 
then obtain  $a$  where  $?p1\ a \wedge ?p2\ a \wedge ?eq\ a$ 
by auto

```

```

let  $?a = \lambda i. a \bmod n$ 
have  $?p1\ ?a \wedge ?p2\ ?a$ 
  using  $\langle ?p1\ a \rangle \langle n \geq 3 \rangle$  n-plus-1-mod-n n-plus-2-mod-n
  by auto

```

```

have eq:  $\forall i. ?a\ i * ?a\ (i + 1) + 1 = ?a\ (i + 2)$ 
proof safe
  fix i
  have a  $((i + 1) \bmod n) = a\ (i \bmod n + 1)$ 
    using  $\langle ?p1\ a \rangle$ 
    by (simp add: mod-Suc)

moreover

have a  $((i + 2) \bmod n) = a\ (i \bmod n + 2)$ 
  using  $\langle ?p1\ a \rangle\ \langle ?p2\ a \rangle$ 
  by (metis One-nat-def Suc-eq-plus1 add-Suc-right mod-Suc one-add-one)

ultimately

show a  $(i \bmod n) * a\ ((i + 1) \bmod n) + 1 = a\ ((i + 2) \bmod n)$ 
  using  $\langle ?eq\ a \rangle$ 
  using assms
  by auto
qed

have *:  $\forall i. ?a\ i > 0 \wedge ?a\ (i + 1) > 0 \longrightarrow ?a\ (i + 2) > 1$ 
  using eq
  by (smt mult-pos-pos)

have no-pos-pos:  $\forall i. \neg (?a\ i > 0 \wedge ?a\ (i + 1) > 0)$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain i where  $?a\ i > 0\ ?a\ (i + 1) > 0$ 
  by auto

have  $\forall j \geq i+1. ?a\ j > 0 \wedge ?a\ (j + 1) > 1$ 
proof (rule allI, rule impI)
  fix j
  assume  $i + 1 \leq j$ 
  then show  $0 < ?a\ j \wedge 1 < ?a\ (j + 1)$ 
  proof (induction j)
    case 0
    then show ?case

```

```

      by simp
    next
      case (Suc j)
      show ?case
      proof (cases  $i + 1 \leq j$ )
        case False
        then have  $i + 1 = \text{Suc } j$ 
          using Suc(2)
          by auto
        then show ?thesis
          using  $\langle ?a \ i > 0 \rangle \langle ?a \ (i + 1) > 0 \rangle *$ 
          by auto
      next
        case True
        then show ?thesis
          using Suc(1) *
          by (smt Suc-eq-plus1 add-Suc-right one-add-one)
      qed
    qed
  qed

  then have  $\forall j \geq i+2. ?a \ j > 1$ 
  by (metis Suc-eq-plus1 add-Suc-right le-iff-add one-add-one plus-nat.simps(2))

  have *:  $\forall j \geq i+2. ?a \ (j + 2) > ?a \ (j + 1)$ 
  proof safe
    fix j
    assume  $i + 2 \leq j$ 
    then have  $?a \ j > 1 \ ?a \ (j + 1) > 1$ 
      using  $\langle \forall j \geq i + 2. ?a \ j > 1 \rangle \langle i + 2 \leq j \rangle$ 
      by auto
    then have  $?a \ (j + 1) < ?a \ j * ?a \ (j + 1)$ 
      by simp
    then show  $?a \ (j + 2) > ?a \ (j + 1)$ 
      using eq
      by smt
  qed

  have  $\forall j > i + 3. ?a \ j > ?a \ (i + 3)$ 
  proof safe

```

```

fix  $j$ 
assume  $i + 3 < j$ 
then show  $a((i + 3) \bmod n) < a(j \bmod n)$ 
proof (induction  $j$ )
  case 0
    then show ?case
    by simp
  next
    case (Suc  $j$ )
    show ?case
    proof (cases  $i + 3 < j$ )
      case True
        then have ?a ( $i + 3$ ) < ?a  $j$ 
        using Suc
        by simp
        also have ?a  $j$  < ?a ( $j + 1$ )
        using Suc(2)
        using *[rule-format, of  $j-1$ ]
        by simp
      finally
        show ?thesis
        by simp
    next
      case False
      then have  $i + 3 = j$ 
      using Suc(2)
      by simp
      then show ?thesis
      using *[rule-format, of  $i+2$ ]
      by (metis One-nat-def Suc-1 Suc-eq-plus1 add-Suc-right less-or-eq-imp-le
numeral-3-eq-3)
    qed
  qed
qed

```

```

then have ?a ( $i + 3 + n$ ) > ?a ( $i + 3$ )
by (meson assms less-add-same-cancel1 less-le-trans zero-less-numeral)

```

moreover

```

have ?a (i + 3 + n) = ?a (i + 3)
  by simp

ultimately

show False
  by simp
qed

have no-zero:  $\forall i. ?a i \neq 0$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain i where ?a i = 0
    by auto
  then have ?a (i + n) = 0
    by auto
  have ?a (i + n + 2) = 1
    using ⟨?a (i + n) = 0⟩ eq
    by (metis add.commute mult-zero-left nat-arith.rule0)
  moreover
  have ?a (i + n + 1) = 1
    using ⟨?a (i + n) = 0⟩ eq[rule-format, of i+n-1] ⟨n ≥ 3⟩
    by simp
  ultimately
  show False
    using no-pos-pos
    by (smt add.assoc one-add-one)
qed

have neg-neg-pos:  $\forall i. ?a i < 0 \wedge ?a (i + 1) < 0 \longrightarrow ?a (i + 2) > 1$ 
  using eq
  by (smt mult-neg-neg)

{
  fix i
  assume ?a i < 0 ?a (i + 1) < 0
  then have ?a (i + 2) > 1
    using neg-neg-pos
    by simp
  then have ?a (i + 3) < 0

```



```

using no-pos-pos no-zero
by (smt One-nat-def Suc-eq-plus1 add-Suc-right numeral-3-eq-3 one-add-one)

have  $?a\ (i + 4) < 1$ 
proof–
  have  $?a\ (i + 4) = ?a\ (i+2) * ?a\ (i+3) + 1$ 
    using eq[rule-format, of i+2]
    by (simp add: numeral-3-eq-3 numeral-Bit0)
  moreover
    have  $?a\ (i+2) * ?a\ (i + 3) < 0$ 
      using  $\langle ?a\ (i + 3) < 0 \rangle \langle ?a\ (i + 2) > 1 \rangle$ 
      by (simp add: mult-pos-neg)
    ultimately
    show ?thesis
    by simp
qed

then have  $?a\ (i + 4) < ?a\ (i + 2)$ 
  using  $\langle ?a\ (i + 2) > 1 \rangle$ 
  by simp

have  $?a\ (i+5) - ?a\ (i+4) = (?a\ (i+3) * ?a\ (i+4) + 1) - (?a\ (i+3) * ?a\ (i+2) + 1)$ 
  using eq
  by (simp add: Groups.mult-ac(2) numeral-eq-Suc)
also have  $\dots = ?a\ (i+3) * (?a\ (i+4) - ?a\ (i+2))$ 
  by (simp add: field-simps)
finally have  $?a\ (i+5) - ?a\ (i+4) > 0$ 
  using  $\langle ?a\ (i + 4) < ?a\ (i + 2) \rangle \langle ?a\ (i + 3) < 0 \rangle$ 
  by (smt mult-neg-neg)
then have  $?a\ (i + 5) > ?a\ (i + 4)$ 
  by auto
then have  $?a\ (i + 4) < 0$ 
  using no-pos-pos no-zero
  by (smt Suc-eq-plus1 add-Suc-right numeral-eq-Suc pred-numeral-simps(3))

have  $?a\ (i+2) > 0 \wedge ?a\ (i+3) < 0 \wedge ?a\ (i+4) < 0$ 
  using  $\langle 1 < a\ ((i + 2) \bmod n) \rangle \langle a\ ((i + 3) \bmod n) < 0 \rangle \langle a\ ((i + 4) \bmod n) < 0 \rangle$ 
  by simp

```

```

} note after-neg-neg = this

have  $\exists i. ?a\ i < 0 \wedge ?a\ (i + 1) < 0$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have alt:  $\forall i. ?a\ i < 0 \longleftrightarrow ?a\ (i + 1) > 0$ 
    using no-zero no-pos-pos
    by smt

  have neg:  $\forall i\ k. ?a\ i < 0 \longrightarrow ?a\ (i + 2*k) < 0$ 
  proof safe
    fix i k
    assume  $?a\ i < 0$ 
    then show  $?a\ (i + 2 * k) < 0$ 
    proof (induction k)
      case 0
      then show ?case
        by simp
    next
      case (Suc k)
      then show ?case
        using alt
        by (smt add.assoc add.commute mult-Suc-right no-zero one-add-one)
    qed
  qed

  have inc:  $\forall i. ?a\ i < 0 \longrightarrow ?a\ i < ?a\ (i + 2)$ 
  proof safe
    fix i
    assume  $?a\ i < 0$ 
    have  $?a\ (i+1) > 0$ 
      using alt
      using  $(?a\ i < 0)$ 
      by blast
    then have  $?a\ (i+2) < 0$ 
      using alt
      by (smt add.assoc no-zero one-add-one)
    then have  $?a\ (i+3) > 0$ 
      using alt
      by (simp add: numeral-3-eq-3)
  
```

```

have ?a i * ?a (i+1) + 1 < ?a (i+1) * ?a (i+2) + 1
  using ⟨?a (i+2) < 0⟩ ⟨?a (i+3) > 0⟩ eq
  by (simp add: numeral-eq-Suc)
then show ?a i < ?a (i + 2)
  using ⟨?a (i + 1) > 0⟩
  by (smt Groups.mult-ac(2) Suc-eq-plus1 add-2-eq-Suc' alt eq mult-less-cancel-left1)
qed

```

```

obtain i where ?a i < 0
  using alt
  by (meson linorder-negE-linordered-idom no-zero)
have ∀ k ≥ 1. ?a i < ?a (i + 2*k)
proof safe
  fix k::nat
  assume 1 ≤ k
  then show ?a i < ?a (i + 2*k)
  proof (induction k)
    case 0
    then show ?case
      by simp
  next
    case (Suc k)
    show ?case
    proof (cases k = 0)
      case True
      then show ?thesis
        using inc ⟨?a i < 0⟩
        by auto
    next
      case False
      then show ?thesis
        using ?a i < 0
        using Suc(1) inc[rule-format, of i + 2*k] neg[rule-format, of i k]
        by simp
    qed
  qed
qed
then have ?a i < ?a (i + 2*n)
  using ⟨n ≥ 3⟩
  by (simp add: numeral-eq-Suc)

```

```

    then show False
      by simp
    qed

  then obtain i where  $?a\ i < 0 \wedge ?a\ (i + 1) < 0$ 
    by auto

  have neg-neg-pos:  $\forall k. ?a\ (i + 3 * k) < 0 \wedge ?a\ (i + 1 + 3*k) < 0 \wedge ?a\ (i + 2 + 3*k) > 0$  (is  $\forall k. ?P\ k$ )
  proof
    fix k
    show  $?P\ k$ 
    proof (induction k)
      case 0
      then show  $?case$ 
        using  $\langle ?a\ i < 0 \rangle \langle ?a\ (i + 1) < 0 \rangle$  after-neg-neg[of i]
        by simp
      next
      case (Suc k)
      then show  $?case$ 
        using after-neg-neg[of i + 3*k]
        using after-neg-neg[of i + 3*k + 3]
        by (simp add: numeral-3-eq-3 numeral-Bit0)
    qed
  qed

show  $3\ dvd\ n$ 
proof-
  have  $n\ mod\ 3 = 0 \vee n\ mod\ 3 = 1 \vee n\ mod\ 3 = 2$ 
    by auto
  then show  $?thesis$ 
  proof
    assume  $n\ mod\ 3 = 0$ 
    then show  $?thesis$ 
      by auto
  next
    assume  $n\ mod\ 3 = 1 \vee n\ mod\ 3 = 2$ 
    then have False
    proof
      assume  $n\ mod\ 3 = 1$ 

```

```

    then obtain  $k$  where  $n = 3 * k + 1$ 
    by (metis add-diff-cancel-left' add-diff-cancel-right' add-eq-if assms dvd-minus-mod
dvd-mult-div-cancel not-numeral-le-zero plus-1-eq-Suc)
    then have  $?a (i + 1) = ?a (i + 2 + 3*k)$ 
    by (metis add.assoc add-Suc-right mod-add-self2 one-add-one plus-1-eq-Suc)
    then show False
      using neg-neg-pos[rule-format, of 0] neg-neg-pos[rule-format, of k]
      by simp
  next
    assume  $n \bmod 3 = 2$ 
    then obtain  $k$  where  $n = 3 * k + 2$ 
    by (metis One-nat-def Suc-1 add commute add-Suc-shift add-diff-cancel-left'
assms dvd-minus-mod dvd-mult-div-cancel le-iff-add numeral-3-eq-3)
    then have  $?a i = ?a (i + 2 + 3*k)$ 
    by (metis add.assoc add-Suc-right mod-add-self2 one-add-one plus-1-eq-Suc)
    then show False
      using neg-neg-pos[rule-format, of 0] neg-neg-pos[rule-format, of k]
      by simp
  qed
then show ?thesis
  by simp
qed
qed
qed
end

```

5.1.2 IMO 2018 SL - A4

```

theory IMO-2018-SL-A4-sol
imports Complex-Main
begin

```

```

definition is-Max :: ' $a::linorder$  set  $\Rightarrow$  ' $a \Rightarrow$  bool' where
  is-Max A  $x \longleftrightarrow x \in A \wedge (\forall x' \in A. x' \leq x)$ 

```

```

lemma sum-list-cong:

```

```

  assumes  $\bigwedge x. x \in \text{set } l \implies f x = g x$ 
  shows  $(\sum x \leftarrow l. f x) = (\sum x \leftarrow l. g x)$ 
  using assms

```

by (*metis map-eq-conv*)

lemma *Max-ge-Min*:

assumes *finite A A* $\neq \{\}$

shows $\text{Max } A \geq \text{Min } A$

using *assms*

by *simp*

theorem *IMO2018SL-A4*:

shows

is-Max $\{a \text{ 2018} - a \text{ 2017} \mid a::\text{nat} \Rightarrow \text{real}. a \text{ 0} = 0 \wedge a \text{ 1} = 1 \wedge (\forall n \geq 2. \exists k. 1 \leq k \wedge k \leq n \wedge a \text{ n} = (\sum i \leftarrow [n-k..<n]. a \text{ i}) / \text{real } k)\}$
 $(2016 / 2017^2)$ (**is** *is-Max* $\{?f \text{ a} \mid a. ?P \text{ a}\} ?m$)

unfolding *is-Max-def*

proof

show $?m \in \{?f \text{ a} \mid a. ?P \text{ a}\}$

proof–

let $?a = (\lambda n. \text{if } n = 0 \text{ then } 0$
 $\text{else if } n < 2017 \text{ then } 1$
 $\text{else if } n = 2017 \text{ then } 1 - 1/2017$
 $\text{else } 1 - 1/2017^2) :: (\text{nat} \Rightarrow \text{real})$

have $?P ?a$

proof *safe*

show $?a \text{ 0} = 0$

by *simp*

next

show $?a \text{ 1} = 1$

by *simp*

next

fix $n::\text{nat}$

assume $2 \leq n$

show $\exists k. 1 \leq k \wedge k \leq n \wedge ?a \text{ n} = (\sum i \leftarrow [n - k..<n]. ?a \text{ i}) / \text{real } k$

proof (*cases n < 2017*)

case *True*

have $[n-1..<n] = [n-1]$

using $\langle n \geq 2 \rangle$

by (*simp add: upt-rec*)

then show *?thesis*

using $\langle n \geq 2 \rangle \langle n < 2017 \rangle$

```

    by (rule-tac x=1 in exI, auto)
next
  case False
  show ?thesis
  proof (cases n = 2017)
    case True
    have [0..<2017] = [0] @ [1..<2017]
    by (metis One-nat-def less-numeral-extra(4) numeral-eq-Suc plus-1-eq-Suc
upt-add-eq-append upt-rec zero-le-one zero-less-one)
    then have  $(\sum i \leftarrow [0..<2017]. ?a\ i) = ?a\ 0 + (\sum i \leftarrow [1..<2017]. ?a\ i)$ 
    by simp
    then have  $(\sum i \leftarrow [0..<2017]. ?a\ i) = (\sum i \leftarrow [0..<1]. 0) + (\sum i \leftarrow [1..<2017].$ 
1)
    using sum-list-cong[of [1..<2017] ?a  $\lambda k. 1$ ]
    by auto
    then have  $(\sum i \leftarrow [0..<2017]. ?a\ i) = 2016$ 
    by (simp add: sum-list-triv)
    then show ?thesis
    using ⟨n = 2017⟩
    by (rule-tac x=2017 in exI, auto)
next
  case False
  show ?thesis
  proof (cases n = 2018)
    case True
    have [1..<2018] = [1..<2017] @ [2017]
    by (metis one-le-numeral one-plus-numeral plus-1-eq-Suc semiring-norm(4)
semiring-norm(5) upt-Suc-append)
    then have  $(\sum i \leftarrow [1..<2018]. ?a\ i) = (\sum i \leftarrow [1..<2017]. ?a\ i) + ?a$ 
2017
    using sum-list-append[of [1..<2017] [2017..<2018]]
    by simp
    then have  $(\sum i \leftarrow [1..<2018]. ?a\ i) = 2016 + (1 - 1/2017)$ 
    using sum-list-cong[of [1..<2017] ?a  $\lambda k. 1$ ]
    by (simp add: sum-list-triv)
    then show ?thesis
    using ⟨n = 2018⟩
    by (rule-tac x=2017 in exI, auto)
next
  case False

```

```

      have  $[n-1..<n] = [n-1]$ 
      using  $\langle n \geq 2 \rangle$ 
      by (simp add: upt-rec)
    then show ?thesis
      using  $\langle \neg n < 2017 \rangle \langle n \neq 2017 \rangle \langle n \neq 2018 \rangle \langle n \geq 2 \rangle$ 
      by (rule-tac x=1 in exI, auto)
  qed
qed
qed
qed
moreover
have  $?f ?a = ?m$ 
  by simp
ultimately
show ?thesis
  by (smt mem-Collect-eq)
qed
next
show  $\forall x' \in \{?f a \mid a. ?P a\}. x' \leq ?m$ 
proof safe
  fix  $a :: nat \Rightarrow real$ 
  let  $?S = \lambda n k. (\sum i \leftarrow [n-k..<n]. a i)$ 
  assume  $a 0 = 0 \wedge a 1 = 1$  and *:  $\forall n \geq 2. \exists k \geq 1. k \leq n \wedge a n = ?S n k / real$ 
 $k$ 
  let  $?A = \lambda n. \{?S n k / k \mid k. k \in \{1..<n+1\}\}$ 
  let  $?max = \lambda n. Max (?A n)$ 
  let  $?min = \lambda n. Min (?A n)$ 
  let  $? \Delta = \lambda n. ?max n - ?min n$ 

  have  $A: \forall n \geq 1. finite (?A n) \wedge ?A n \neq \{\}$ 
  by auto

  have  $\forall n \geq 2. ? \Delta n \geq 0$ 
proof safe
  fix  $n :: nat$ 
  assume  $2 \leq n$ 
  then have  $?max n \geq ?min n$ 
    using Max-ge-Min[of ?A n] A[rule-format, of n]
    by force
  then show  $? \Delta n \geq 0$ 

```


by *simp*
qed

have $\forall n \geq 2. ?min\ n \leq a\ n \wedge a\ n \leq ?max\ n$
proof *safe*
 fix $n::nat$
 assume $n \geq 2$
 then have $n \geq 1$
 by *simp*
 have $a\ n \in ?A\ n$
 using $*\ \langle n \geq 2 \rangle$
 by *force*
 then show $?min\ n \leq a\ n \wedge a\ n \leq ?max\ n$
 using $A[rule-format, OF\ \langle n \geq 1 \rangle]$
 using $Min-le[of\ ?A\ n\ a\ n]\ Max-ge[of\ ?A\ n\ a\ n]$
 by *blast+*
 qed

have $\forall n \geq 2. a\ (n - 1) \in ?A\ n$
proof *safe*
 fix $n::nat$
 assume $n \geq 2$
 then have $[n-1..<n] = [n-1]$
 using *upt-rec* by *auto*
 then have $a\ (n - 1) = ?S\ n\ 1$
 by *simp*
 then show $\exists k. a\ (n - 1) = ?S\ n\ k / k \wedge k \in \{1..<n+1\}$
 using $\langle n \geq 2 \rangle$
 by *force*
 qed

have $\forall n \geq 2. ?min\ n \leq a\ (n-1) \wedge a\ (n-1) \leq ?max\ n$
proof *safe*
 fix $n::nat$
 assume $n \geq 2$
 then have $n \geq 1$
 by *simp*
 have $a\ (n - 1) \in ?A\ n$
 using $\langle \forall n \geq 2. a\ (n - 1) \in ?A\ n \rangle\ \langle n \geq 2 \rangle$
 by *force*

```

then show ?min  $n \leq a (n - 1)$   $a (n - 1) \leq$  ?max  $n$ 
  using  $A[\text{rule-format}, OF \langle n \geq 1 \rangle]$ 
  using  $Min-le[\text{of } ?A \ n \ a \ (n - 1)] \ Max-ge[\text{of } ?A \ n \ a \ (n - 1)]$ 
  by blast+
qed

have ?f  $a \leq$  ? $\Delta$  2018
  using  $\langle \forall \ n \geq 2. \ ?min \ n \leq a \ n \wedge a \ n \leq \ ?max \ n \rangle[\text{rule-format}, \text{of } 2018]$ 
  using  $\langle \forall \ n \geq 2. \ ?min \ n \leq a \ (n-1) \wedge a \ (n-1) \leq \ ?max \ n \rangle[\text{rule-format}, \text{of } 2018]$ 
  by auto

have Claim1:  $\forall \ n > 2. \ ?\Delta \ n \leq (n-1)/n * \ ?\Delta \ (n-1)$ 
proof safe
  fix  $n::nat$ 
  assume  $2 < n$ 
  then have  $1 \leq n$ 
    by simp
  obtain  $k$  where  $?max \ n = ?S \ n \ k / k \ 1 \leq k \ k \leq n$ 
    using  $A[\text{rule-format}, OF \langle 1 \leq n \rangle] \ Max-in[\text{of } ?A \ n]$ 
    by force
  obtain  $l$  where  $?min \ n = ?S \ n \ l / l \ 1 \leq l \ l \leq n$ 
    using  $A[\text{rule-format}, OF \langle 1 \leq n \rangle] \ Min-in[\text{of } ?A \ n]$ 
    by force

  have  $[n - k..<n] = [n - 1 - (k - 1)..<n - 1] @ [n - 1]$ 
    using  $\langle 1 \leq k \rangle \langle k \leq n \rangle \langle 1 \leq n \rangle$ 
  by (metis Nat.diff-diff-eq diff-le-self le-add-diff-inverse plus-1-eq-Suc upt-Suc-append)
  then have  $?S \ n \ k = ?S \ (n-1) \ (k-1) + a \ (n-1)$ 
    by simp

  have  $[n - l..<n] = [n - 1 - (l - 1)..<n - 1] @ [n - 1]$ 
    using  $\langle 1 \leq l \rangle \langle l \leq n \rangle \langle 1 \leq n \rangle$ 
  by (metis Nat.diff-diff-eq diff-le-self le-add-diff-inverse plus-1-eq-Suc upt-Suc-append)
  then have  $?S \ n \ l = ?S \ (n-1) \ (l-1) + a \ (n-1)$ 
    by simp

  have real  $(k - Suc \ 0) = \text{real } k - 1$ 
    using  $\langle k \geq 1 \rangle$ 
    by simp

```

```

have ?S (n-1) (k-1) ≤ (k - 1) * ?max (n - 1)
proof (cases k = 1)
  case True
  then show ?thesis
    by simp
next
  case False
  have n-1 ≥ 1
    using ⟨n > 2⟩
    by simp
  have ?S (n-1) (k-1) / (k - 1) ≤ ?max (n - 1)
proof (rule Max-ge)
  show finite (?A (n-1))
    using A[rule-format, OF ⟨n-1 ≥ 1⟩]
    by simp
next
  show ?S (n-1) (k-1) / (k - 1) ∈ ?A (n-1)
    using ⟨k ≠ 1⟩ ⟨k ≥ 1⟩ ⟨k ≤ n⟩
    by simp (rule-tac x=k-1 in exI, auto)
qed
then show ?thesis
  using ⟨k ≥ 1⟩ ⟨k ≠ 1⟩
  by (simp add: field-simps)
qed

have ?S (n-1) (l-1) ≥ (l - 1) * ?min (n - 1)
proof (cases l = 1)
  case True
  then show ?thesis
    by simp
next
  case False
  have n-1 ≥ 1
    using ⟨n > 2⟩
    by simp
  have ?S (n-1) (l-1) / (l - 1) ≥ ?min (n - 1)
proof (rule Min-le)
  show finite (?A (n-1))
    using A[rule-format, OF ⟨n-1 ≥ 1⟩]

```

```

      by simp
    next
      show ?S (n-1) (l-1) / (l - 1) ∈ ?A (n-1)
        using ⟨l ≠ 1⟩ ⟨l ≥ 1⟩ ⟨l ≤ n⟩
        by simp (rule-tac x=l-1 in exI, auto)
    qed
  then show ?thesis
    using ⟨l ≥ 1⟩ ⟨l ≠ 1⟩
    by (simp add: field-simps)
  qed

  have ?min (n-1) ≤ a (n-1) a (n-1) ≤ ?max (n-1)
    using ⟨∀ n ≥ 2. ?min n ≤ a n ∧ a n ≤ ?max n⟩[rule-format, of n-1] ⟨n
> 2⟩
    by simp-all

  {
    fix x1 x2::real
    assume 0 < x1 x1 ≤ x2
    then have (x1 - 1) / x1 ≤ (x2 - 1) / x2
      by (metis (no-types, hide-lams) diff-divide-distrib diff-mono divide-self-if
frac-le leD order-refl zero-le-one)
    } note mono = this

  have k*(?max n - a (n-1)) = ?S n k - k * a (n-1)
    using ⟨?max n = ?S n k / k⟩
    by (simp add: algebra-simps)
  also have ... = ?S (n-1) (k-1) - (real k - 1) * a (n-1)
    using ⟨?S n k = ?S (n-1) (k-1) + a (n-1)⟩
    by (simp add: field-simps)
  also have ... ≤ (k - 1) * ?max (n - 1) - (real k - 1) * a (n-1)
    using ⟨?S (n-1) (k-1) ≤ (k - 1) * ?max (n - 1)⟩
    by simp
  also have ... = (real k - 1) * (?max (n - 1) - a (n-1))
    using ⟨k ≥ 1⟩
    by (auto simp add: right-diff-distrib)
  finally have k*(?max n - a (n-1)) ≤ (real k - 1) * (?max (n - 1) - a
(n-1))
    .

```

then have $?max\ n - a\ (n-1) \leq (real\ k - 1) / k * (?max\ (n-1) - a\ (n-1))$

using $\langle k \geq 1 \rangle$

by (*simp add: field-simps*)

also have $(real\ k - 1) / k * (?max\ (n-1) - a\ (n-1)) \leq$
 $(real\ n - 1) / n * (?max\ (n-1) - a\ (n-1))$

proof—

have $(real\ k - 1) / k \leq (real\ n - 1) / n$

using *mono[*of real k real n*] $\langle k \leq n \rangle \langle k \geq 1 \rangle$*

by *simp*

then show *?thesis*

using $\langle a\ (n - 1) \leq ?max\ (n-1) \rangle$

by (*smt mult-cancel-right real-mult-le-cancel-iff1*)

qed

finally

have $1: ?max\ n - a\ (n-1) \leq (real\ n - 1) / n * (?max\ (n-1) - a\ (n-1))$

.

have $l * (a\ (n-1) - ?min\ n) = l * a\ (n-1) - ?S\ n\ l$

using $\langle ?min\ n = ?S\ n\ l / l \rangle$

by (*simp add: algebra-simps*)

also have $\dots = (real\ l - 1) * a\ (n-1) - ?S\ (n-1)\ (l-1)$

using $\langle ?S\ n\ l = ?S\ (n-1)\ (l-1) + a\ (n-1) \rangle$

by (*simp add: field-simps*)

also have $\dots \leq (real\ l - 1) * a\ (n-1) - (l - 1) * ?min\ (n - 1)$

using $\langle ?S\ (n-1)\ (l-1) \geq (l - 1) * ?min\ (n - 1) \rangle$

by (*simp add: field-simps*)

also have $\dots = (real\ l - 1) * (a\ (n-1) - ?min\ (n - 1))$

using $\langle l \geq 1 \rangle$

by (*auto simp add: right-diff-distrib*)

finally have $l * (a\ (n-1) - ?min\ n) \leq (real\ l - 1) * (a\ (n-1) - ?min\ (n - 1))$
 $- 1))$

.

then have $a\ (n-1) - ?min\ n \leq (real\ l - 1) / l * (a\ (n-1) - ?min\ (n-1))$

using $\langle l \geq 1 \rangle$

by (*simp add: field-simps*)

also have $(real\ l - 1) / l * (a\ (n-1) - ?min\ (n-1)) \leq$
 $(real\ n - 1) / n * (a\ (n-1) - ?min\ (n-1))$

proof—

have $(real\ l - 1) / l \leq (real\ n - 1) / n$

```

    using mono[of real l real n] ⟨l ≤ n⟩ ⟨l ≥ 1⟩
    by simp
  then show ?thesis
    using ⟨a (n - 1) ≥ ?min (n-1)⟩
    by (smt mult-cancel-right real-mult-le-cancel-iff1)
qed
finally
have 2: a (n-1) - ?min n ≤ (real n - 1) / n * (a (n-1) - ?min (n-1))
.

have ?Δ n = (?max n - a (n-1)) + (a (n-1) - ?min n)
  by simp
also have ... ≤ (real n - 1) / n * ((?max (n-1) - a (n-1)) + (a (n-1)
- ?min (n-1)))
  using 1 2
  by (simp add: right-diff-distrib')
finally show ?Δ n ≤ (real n - 1) / n * ?Δ (n-1)
  by simp
qed

obtain Δ where Δ = ?Δ by auto
then have Claim1': ∀ n > 2. Δ n ≤ (n-1)/n * Δ (n-1)
  using Claim1
  by blast

have Claim1-iter': ∧ N q. [2 ≤ q; q ≤ N] ⇒ Δ (N+1) ≤ Δ (q+1) * (q +
1) / (N + 1)
proof-
  fix N q :: nat
  assume 2 ≤ q q ≤ N
  then show Δ (N+1) ≤ Δ (q+1) * (q + 1) / (N + 1)
  proof (induction N)
    case 0
    then show ?case
      by simp
  next
    case (Suc N)
    show ?case
    proof (cases q ≤ N)
      case True

```

```

    have  $\Delta (N + 2) \leq ((N + 1)/(N + 2)) * \Delta (N + 1)$ 
      using Claim1'[rule-format, of Suc N + 1]  $\langle 2 \leq q \rangle \langle q \leq N \rangle$ 
      by simp
    moreover
    have  $\Delta (N + 1) \leq \Delta (q + 1) * (q + 1) / (N + 1)$ 
      using True  $\langle 2 \leq q \rangle$  Suc(1)
      by simp
    then have  $((N + 1)/(N + 2)) * \Delta (N + 1) \leq ((N + 1)/(N + 2)) * (\Delta (q + 1) * (q + 1) / (N + 1))$ 
      by (subst real-mult-le-cancel-iff2, simp-all)
    ultimately
    show ?thesis
      by simp
  next
  case False
  then have  $q = N + 1$ 
    using Suc(3)
    by simp
  then show ?thesis
    by simp
qed
qed
qed

{
  fix  $q::nat$ 
  assume  $\forall n. 1 \leq n \wedge n < q \longrightarrow a\ n = 1$ 

  have  $\forall k. 1 \leq k \wedge k < q \longrightarrow ?S\ q\ k = k$ 
  proof safe
    fix  $k::nat$ 
    assume  $1 \leq k \wedge k < q$ 
    then have  $(\sum i \leftarrow [q-k..<q]. a\ i) = (\sum i \leftarrow [q-k..<q]. 1)$ 
      using sum-list-cong[of  $[q-k..<q]$   $a\ \lambda i. 1$ ]
      using  $\langle \forall n. 1 \leq n \wedge n < q \longrightarrow a\ n = 1 \rangle \langle k < q \rangle$ 
      by fastforce
    then show  $?S\ q\ k = k$ 
      using  $\langle 1 \leq k \rangle \langle k < q \rangle$ 
      by (simp add: sum-list-triv)
  qed
}

```

```

}
note all-1-Sqk = this

{
  fix q::nat
  assume  $q \geq 2$ 
  assume  $\forall n. 1 \leq n \wedge n < q \longrightarrow a\ n = 1$ 
  have  $?S\ q\ q = q - 1$ 
  proof–
    have  $[q - q..<q] = [0] @ [1..<q]$ 
    using  $\langle 2 \leq q \rangle$ 
    using upt-rec by auto
    then have  $?S\ q\ q = (\sum\ i \leftarrow [1..<q].\ a\ i)$ 
    using  $\langle a\ 0 = 0 \rangle$ 
    by auto
    also have  $\dots = (\sum\ i \leftarrow [1..<q].\ 1::real)$ 
    using sum-list-cong[of  $[1..<q]\ a\ \lambda\ i.\ 1$ ]
    using  $\langle \forall n. 1 \leq n \wedge n < q \longrightarrow a\ n = 1 \rangle$ 
    by simp
    finally show ?thesis
    by (simp add: sum-list-triv)
  qed
} note all-1-Sqq = this

show  $?f\ a \leq ?m$ 
proof (cases  $\forall n. 2 \leq n \wedge n \leq 2017 \longrightarrow a\ n = 1$ )
  case True
  then have  $\forall n. 1 \leq n \wedge n < 2018 \longrightarrow a\ n = 1$ 
  using  $\langle a\ 1 = 1 \rangle$ 
  by (metis Suc-leI add-le-cancel-left le-eq-less-or-eq one-add-one one-plus-numeral
plus-1-eq-Suc semiring-norm(4) semiring-norm(5))
  then have  $\forall k. 1 \leq k \wedge k \leq 2018 \longrightarrow ?S\ 2018\ k \leq k$ 
  using all-1-Sqk[of 2018] all-1-Sqq[of 2018]
  by (smt Suc-leI le-eq-less-or-eq of-nat-1 of-nat-diff one-add-one one-less-numeral-iff
plus-1-eq-Suc semiring-norm(76))
  then have  $a\ 2018 \leq 1$ 
  using  $*[rule-format, of\ 2018]$ 
  by auto
then show ?thesis
using True

```



```

    by auto
next
  case False
  let ?Q = {q. 2 ≤ q ∧ q ≤ 2017 ∧ a q ≠ 1}
  let ?q = Min ?Q
  have ?Q ≠ {}
    using False ⟨a 1 = 1⟩
    by auto
  then have 2 ≤ ?q ?q ≤ 2017 a ?q ≠ 1
    using Min-in[of ?Q]
    by auto

  have ∀ n. 2 ≤ n ∧ n < ?q ⟶ a n = 1
  proof (rule ccontr)
    assume ¬ ?thesis
    then obtain n where 2 ≤ n n < ?q a n ≠ 1
      by auto
    then have n ∈ ?Q
      using ⟨?q ≤ 2017⟩
      by auto
    then show False
      using Min-le[of ?Q n] ⟨?Q ≠ {}⟩ ⟨a n ≠ 1⟩ ⟨n < ?q⟩
      by auto
  qed

  obtain q where q = ?q 2 ≤ q q ≤ 2017 using ⟨2 ≤ ?q⟩ ⟨?q ≤ 2017⟩ by
auto
  then have ∀ n. 1 ≤ n ∧ n < q ⟶ a n = 1
    using ⟨∀ n. 2 ≤ n ∧ n < ?q ⟶ a n = 1⟩ ⟨a 1 = 1⟩
    by (metis Suc-1 Suc-leI le-eq-less-or-eq)
  then have ∀ k. 1 ≤ k ∧ k < q ⟶ ?S q k = k ?S q q = q - 1
    using all-1-Sqk[of q] all-1-Sqq[of q] ⟨2 ≤ q⟩
    by simp-all
  then have ∀ k. 1 ≤ k ∧ k ≤ q ⟶ ?S q k ≤ k
    using le-eq-less-or-eq
    by auto
  then have a q ≤ 1
    using *[rule-format, OF ⟨2 ≤ q⟩]
    by auto
  then have a q < 1

```

```

using  $\langle q = ?q \rangle \langle a \ ?q \neq 1 \rangle$ 
by auto

have  $a \ q = ?S \ q \ q / q$ 
using  $*[rule-format, OF \ \langle 2 \leq q \rangle] \langle a \ q < 1 \rangle \langle \forall \ k. \ 1 \leq k \wedge k < q \longrightarrow ?S \ q$ 
 $k = k \rangle$ 
by (metis div-by-1 less-le of-nat-1 of-nat-le-iff one-eq-divide-iff order-class.order.antisym
zero-le-one)

then have  $a \ q = 1 - 1/q$ 
using  $\langle ?S \ q \ q = q - 1 \rangle$ 
using  $\langle q \geq 2 \rangle$ 
by (simp add: field-simps)

have  $\forall \ i. \ 1 \leq i \wedge i \leq q \longrightarrow ?S \ (q+1) \ i = i - 1/q$ 
proof safe
  fix  $i$ 
  assume  $1 \leq i \wedge i \leq q$ 
  show  $?S \ (q+1) \ i = i - 1/q$ 
  proof (cases i = 1)
    case True
    then show ?thesis
      using  $\langle a \ q = 1 - 1/q \rangle$ 
      by simp
  next
  case False
  then have  $?S \ (q+1) \ i = a \ q + ?S \ q \ (i-1)$ 
    using  $\langle 1 \leq i \rangle \langle i \leq q \rangle$ 
    by auto
  moreover
  have  $?S \ q \ (i-1) = (i-1)$ 
    using  $\langle \forall \ k. \ 1 \leq k \wedge k < q \longrightarrow ?S \ q \ k = k \rangle [rule-format, of \ i-1]$ 
    using  $\langle 1 \leq i \rangle \langle i \leq q \rangle \langle i \neq 1 \rangle$ 
    using Suc-le-eq
    by auto
  ultimately
  show ?thesis
    using  $\langle a \ q = 1 - 1/q \rangle \langle 1 \leq i \rangle$ 
    by simp
qed

```

```

qed

have ?S (q+1) (q+1) = q - 1/q
proof-
  have ?S (q+1) (q+1) = a q + ?S q q
    by simp
  then show ?thesis
    using ⟨?S q q = q - 1⟩ ⟨a q = 1 - 1/q⟩
    using ⟨2 ≤ q⟩
    by simp
qed

have qq: (real q - 1 / real q) / (real q + 1) = (real q - 1) / real q
proof-
  have (real q + 1) * ((real q - 1 / real q) / (real q + 1)) = (real q + 1) *
((real q - 1) / real q)
    using ⟨2 ≤ q⟩
    by simp (simp add: field-simps)
  then show ?thesis
    by (subst (asm) mult-left-cancel, simp-all)
qed

have ?min (q+1) = (real q - 1)/real q (is ?lhs = ?mn)
proof (subst Min-eq-iff)
  show finite (?A (q+1))
    by simp
next
  show ?A (q+1) ≠ {}
    using ⟨q ≥ 2⟩
    by auto
next
  show ?mn ∈ ?A (q+1) ∧ (∀ m' ∈ ?A (q+1). m' ≥ ?mn)
proof
  have ?mn = 1 - 1/q
    using ⟨2 ≤ q⟩
    by (simp add: field-simps)
  then have ?mn = ?S (q+1) 1
    using ⟨∀ i. 1 ≤ i ∧ i ≤ q ⟶ ?S (q+1) i = i - 1/q⟩[rule-format, of
1] ⟨2 ≤ q⟩
    by simp

```

```

then show ?mn ∈ ?A (q+1)
  by force
show ∀ m' ∈ ?A (q+1). m' ≥ ?mn
proof
  fix m'
  assume m' ∈ ?A (q+1)
  then obtain k where k ∈ {1.. $q+1+1$ } m' = ?S (q+1) k / k
    by force
  show m' ≥ ?mn
  proof (cases k ≤ q)
    case True
      then have m' = (k - 1/q) / k
        using ⟨k ∈ {1.. $q+1+1$ }⟩ ⟨m' = ?S (q+1) k / k⟩
        using ⟨∀ i. 1 ≤ i ∧ i ≤ q ⟶ ?S (q+1) i = i - 1/q⟩
        by auto
      then have m' = 1 - 1/(q*k)
        using ⟨k ∈ {1.. $q+1+1$ }⟩ ⟨q ≥ 2⟩
        by (simp add: field-simps)
      then show ?thesis
        using ⟨?mn = 1 - 1/q⟩ ⟨k ∈ {1.. $q+1+1$ }⟩ ⟨2 ≤ q⟩
        by simp (simp add: field-simps)
    next
      case False
      then have k = q+1
        using ⟨k ∈ {1.. $q+1+1$ }⟩
        by simp
      then have m' = (real q - 1) / real q
        using ⟨m' = ?S (q+1) k / k⟩ ⟨?S (q+1) (q+1) = q - 1/q⟩
        using qq
        by (metis of-nat-1 of-nat-add)
      then show ?thesis
        by simp
    qed
  qed
qed
qed
moreover

have ?max (q+1) = ((real q) ^ 2 - 1) / (real q) ^ 2 (is ?lhs = ?mx)

```

```

proof (subst Max-eq-iff)
  show finite (?A (q+1))
    by simp
next
  show ?A (q+1) ≠ {}
    using ⟨q ≥ 2⟩
    by auto
next
  show ?mx ∈ ?A (q+1) ∧ (∀ m' ∈ ?A (q+1). m' ≤ ?mx)
  proof
    have ?mx = (?S (q+1) q) / q
      using ⟨∀ i. 1 ≤ i ∧ i ≤ q ⟶ ?S (q+1) i = i - 1/q⟩[rule-format, of
q] ⟨2 ≤ q⟩
      by simp (simp add: field-simps power2-eq-square)
    moreover
      have q ∈ {1..<q + 1 + 1}
        using ⟨q ≥ 2⟩
        by simp
      ultimately
      show ?mx ∈ ?A (q+1)
        by force

  show ∀ m' ∈ ?A (q+1). m' ≤ ?mx
  proof
    fix m'
    assume m' ∈ ?A (q+1)
    then obtain k where k ∈ {1..<q+1+1} m' = ?S (q+1) k / k
      by force
    show m' ≤ ?mx
    proof (cases k ≤ q)
      case True
        then have m' = (k - 1/q) / k
          using ⟨k ∈ {1..<q+1+1}⟩ ⟨m' = ?S (q+1) k / k⟩
          using ⟨∀ i. 1 ≤ i ∧ i ≤ q ⟶ ?S (q+1) i = i - 1/q⟩
          by auto
        then have m' = 1 - 1/(q*k)
          using ⟨k ∈ {1..<q+1+1}⟩ ⟨q ≥ 2⟩
          by (simp add: field-simps)
        moreover
          have ?mx = 1 - 1/(q*q)

```

```

    using ⟨q ≥ 2⟩
    by (simp add: field-simps power2-eq-square)
ultimately
show ?thesis
    using ⟨k ≤ q⟩ ⟨2 ≤ q⟩ ⟨k ∈ {1..<q+1+1}⟩
    by simp (simp add: field-simps)
next
case False
then have k = q+1
    using ⟨k ∈ {1..<q+1+1}⟩
    by simp
then have m' = (real q - 1) / real q
    using ⟨m' = ?S (q+1) k / k⟩ ⟨?S (q+1) (q+1) = q - 1/q⟩ qq
    by (metis of-nat-1 of-nat-add)
moreover
have q ≤ q^2
    by (simp add: ⟨2 ≤ q⟩ power2-nat-le-imp-le)
ultimately
show ?thesis
    using ⟨2 ≤ q⟩
    by simp (simp add: field-simps)
qed
qed
qed
qed

ultimately

have ?Δ (q+1) = ((real q)^2 - 1)/(real q)^2 - (real q - 1)/real q
    by simp
also have ... = (real q - 1)/(real q)^2
    using ⟨q ≥ 2⟩
    by (simp add: power2-eq-square field-simps)
finally have del: Δ (q+1) = (real q - 1)/(real q)^2
    using ⟨Δ = ?Δ⟩
    by simp
then have Δ (2017 + 1) ≤ (real q - 1) / (real q)^2 * real (q + 1) / 2018
    using Claim1-iter'[OF ⟨2 ≤ q⟩ ⟨q ≤ 2017⟩]
    by simp
also have ... = ((real q^2 - 1) / (real q)^2) / 2018

```

```

    by (simp add: field-simps power2-eq-square)
  also have ... = (1 - (1 / (real q)2)) / 2018
    using ⟨q ≥ 2⟩
    by (simp add: field-simps)
  also have ... ≤ (1 - (1 / 20172)) / 2018
  proof-
    have q2 ≤ 20172
      using ⟨2 ≤ q⟩ ⟨q ≤ 2017⟩
      using power-mono by blast
    then have (real q)2 ≤ 20172
      by (metis of-nat-le-iff of-nat-numeral of-nat-power)
    then show ?thesis
      using ⟨2 ≤ q⟩
      by (simp add: field-simps power2-eq-square)
  qed
  finally have Δ 2018 ≤ ?m
    by simp

  then show ?thesis
    using ⟨?f a ≤ ?Δ 2018⟩ ⟨Δ = ?Δ⟩
    by simp
  qed
qed
qed
end

```

5.2 Combinatorics problems

5.2.1 IMO 2018 SL - C1

```

theory IMO-2018-SL-C1-sol
imports Complex-Main
begin

```

```

lemma sum-geom-nat:
  fixes q::nat
  assumes q > 1
  shows (∑ k∈{0..n}. qk) = (qn - 1) div (q - 1)
proof (induction n)

```

```

case 0
then show ?case by simp
next
case (Suc n)
then show ?case
  by (smt Nat.add-diff-assoc2 One-nat-def Suc-1 Suc-leI add.commute assms
    div-mult-self4 le-trans mult-eq-if nat-one-le-power one-le-numeral power.simps(2)
    sum.op-ivl-Suc zero-less-diff zero-order(3))
qed

```

```

declare [[smt-timeout = 20]]

```

```

lemma div-diff-nat:
  fixes a b c :: nat
  assumes c dvd a c dvd b
  shows (a - b) div c = a div c - b div c
  using assms
  by (smt add-diff-cancel-left' div-add dvd-diff-nat le-iff-add nat-less-le neq0-conv
    not-less zero-less-diff)

```

```

lemma sum-geom-nat':
  fixes q::nat
  assumes q > 1 m ≤ n
  shows (∑ k∈{m..using assms
proof (induction n)
  case 0
  then show ?case
    by simp
next
case (Suc n)
show ?case
proof (cases m ≤ n)
  case True
  then have sum ((^) q) {m..using Suc
    by simp
  also have ... = ((q^n - q^m) + (q - 1) * q^n) div (q - 1)
    using ⟨q > 1⟩
    by auto

```



```

also have ... = ((q ^ n - q ^ m) + (q^(n+1) - q^n)) div (q - 1)
  by (simp add: algebra-simps)
also have ... = (q ^ (n+1) - q ^ m) div (q - 1)
  using True assms(1) by auto
finally show ?thesis
  by simp
next
case False
then have m = n + 1
  using Suc(3)
  by auto
then show ?thesis
  by simp
qed
qed

```

theorem IMO2018SL-C1:

```

fixes n :: nat
assumes n ≥ 3
shows ∃ (S::nat set). card S = 2*n ∧ (∀ x ∈ S. x > 0) ∧
  (∀ m. 2 ≤ m ∧ m ≤ n → (∃ S1 S2. S1 ∩ S2 = {} ∧ S1 ∪ S2 = S ∧
    card S1 = m ∧ ∑ S1 = ∑ S2))
proof-
  let ?Sa = {(3::nat)^k | k. k ∈ {1..<n}} and ?Sb = {2 * (3::nat)^k | k. k ∈
    {1..<n}} and ?Sc = {1::nat, (3^n + 9) div 2 - 1}
  let ?S = ?Sa ∪ ?Sb ∪ ?Sc

```

```

have finite ?Sa finite ?Sb finite ?Sc finite (?Sa ∪ ?Sb)
  by auto

```

```

have ?Sa ∩ ?Sb = {}

```

proof safe

fix ka kb

assume ka ∈ {1..<n} kb ∈ {1..<n} (3::nat)^ka = 2*3^kb

have odd ((3::nat)^ka) even (2*3^kb)

by simp-all

then have False

using ((3::nat)^ka = 2*3^kb)

by simp

```

    then show  $3^ka \in \{\}$ 
      by simp
    qed

  have  $1 < ((3::nat) ^ n + 9) \text{ div } 2$ 
    by linarith

  have  $\neg 3 \text{ dvd } (((3::nat) ^ n + 9) \text{ div } 2 - 1)$ 
  proof-
    have  $3 \text{ dvd } ((3::nat) ^ n + 9) \text{ div } 2$ 
    proof-
      have  $(3::nat) ^ n + 9 = (3^2) * (3::nat) ^{(n-2)} + 9$ 
      using  $\langle n \geq 3 \rangle$ 
      by (metis One-nat-def add-leD2 le-add-diff-inverse numeral-3-eq-3 one-add-one
        plus-1-eq-Suc power-add)
      then have  $(3::nat) ^ n + 9 = 9 * (3^{(n-2)} + 1)$ 
      by simp
      then have  $((3::nat) ^ n + 9) \text{ div } 2 = (9 * (3^{(n-2)} + 1)) \text{ div } 2$ 
      by auto
      then have  $((3::nat) ^ n + 9) \text{ div } 2 = 9 * ((3^{(n-2)} + 1) \text{ div } 2)$ 
      by (metis One-nat-def div-mult-swap dvd-mult-div-cancel even-add even-power
        even-succ-div-two num.distinct(1) numeral-3-eq-3 numeral-eq-one-iff one-add-one
        plus-1-eq-Suc)
      then show ?thesis
      by simp
    qed
    then show ?thesis
      using  $\langle ((3::nat) ^ n + 9) \text{ div } 2 > 1 \rangle$ 
      by (meson dvd-diffD1 less-imp-le-nat nat-dvd-1-iff-1 numeral-eq-one-iff semiring-norm(86))
    qed

  have  $(?Sa \cup ?Sb) \cap ?Sc = \{\}$ 
  proof-
    have  $?Sa \cap ?Sc = \{\}$ 
    proof safe
      fix k
      assume  $k \in \{1..<n\} \ (3::nat) ^ k = 1$ 
      then show  $3 ^ k \in \{\}$ 
      by simp
    next

```

```

fix  $k$ 
assume  $k \in \{1..<n\} \ (3::nat) \wedge k = (3 \wedge n + 9) \text{ div } 2 - 1$ 
moreover
have  $3 \text{ dvd } (3::nat) \wedge k$ 
  using  $\langle k \in \{1..<n\} \rangle$ 
  by auto
ultimately
have False
  using  $\langle \neg 3 \text{ dvd } (3 \wedge n + 9) \text{ div } 2 - 1 \rangle$ 
  by simp
then show  $3 \wedge k \in \{\}$ 
  by simp
qed

moreover

have  $?Sb \cap ?Sc = \{\}$ 
proof safe
  fix  $k$ 
  assume  $k \in \{1..<n\} \ 2 * (3::nat) \wedge k = 1$ 
  then show  $2 * 3 \wedge k \in \{\}$ 
    by simp
next
  fix  $k$ 
  assume  $k \in \{1..<n\} \ 2 * (3::nat) \wedge k = (3 \wedge n + 9) \text{ div } 2 - 1$ 
  moreover
  have  $3 \text{ dvd } 2 * (3::nat) \wedge k$ 
    using  $\langle k \in \{1..<n\} \rangle$ 
    by auto
  ultimately
  have False
    using  $\langle \neg 3 \text{ dvd } (3 \wedge n + 9) \text{ div } 2 - 1 \rangle$ 
    by simp
  then show  $2 * 3 \wedge k \in \{\}$ 
    by simp
qed

ultimately
show ?thesis
  by blast

```

qed

show *?thesis*

proof (rule-tac $x=?S$ in *exI*, *safe*)

show $\text{card } ?S = 2*n$

proof–

have $\text{card } (?Sa \cup ?Sb) = (n - 1) + (n - 1)$

proof–

have $\text{inj-on } ((\wedge) (\mathcal{I}::\text{nat})) \{1..<n\}$

unfolding *inj-on-def*

by *auto*

then have $\text{card } ?Sa = n-1$

using $\text{card-image}[of \lambda k. \mathcal{I}^k \{1..<n\}]$

by (*smt Collect-cong Setcompr-eq-image card-atLeastLessThan*)

moreover

have $\text{inj-on } (\lambda k. 2 * (\mathcal{I}::\text{nat}) \wedge k) \{1..<n\}$

unfolding *inj-on-def*

by *auto*

then have $\text{card } ?Sb = n-1$

using $\text{card-image}[of \lambda k. 2 * \mathcal{I}^k \{1..<n\}]$

by (*smt Collect-cong Setcompr-eq-image card-atLeastLessThan*)

ultimately

show *?thesis*

using $\langle n \geq 3 \rangle \text{card-Un-disjoint}[of ?Sa ?Sb] \langle ?Sa \cap ?Sb = \{\} \rangle \langle \text{finite } ?Sa \rangle$
 $\langle \text{finite } ?Sb \rangle$

by *smt*

qed

moreover

have $\text{card } \{1, ((\mathcal{I}::\text{nat})^n + 9) \text{ div } 2 - 1\} = 2$

using $\langle 1 < ((\mathcal{I}::\text{nat})^n + 9) \text{ div } 2 \rangle$

by *auto*

ultimately

```

show card ?S = 2*n
  using ⟨n ≥ 3⟩ card-Un-disjoint[of ?Sa ∪ ?Sb ?Sc] ⟨(?Sa ∪ ?Sb) ∩ ?Sc =
  {}⟩ ⟨finite (?Sa ∪ ?Sb)⟩ ⟨finite ?Sc⟩
    by (smt Nat.add-diff-assoc2 Suc-1 Suc-eq-plus1 add-Suc-right card-infinite
  diff-add-inverse2 le-trans mult-2 nat.simps(3) one-le-numeral)
  qed
next
  fix k
  assume k ∈ {1.. $n$ }
  then show 0 < (3::nat) ^ k 0 < 2 * (3::nat) ^ k
    by simp-all
next
  show 0 < ((3::nat) ^ n + 9) div 2 - 1
    using ⟨1 < (3 ^ n + 9) div 2⟩ zero-less-diff
    by blast
next
  fix m
  assume 2 ≤ m m ≤ n
  let ?Am' = {2 * (3::nat) ^ k | k. k ∈ {n-m+1.. $n$ }} and ?Am'' = {(3::nat)
  ^ (n-m+1)}
  let ?Am = ?Am' ∪ ?Am''
  let ?Bm = ?S - ?Am

  have ?Am' ⊆ ?Sb
    using ⟨m ≤ n⟩
    by auto

  have ?Am'' ⊆ ?Sa
    using ⟨m ≤ n⟩ ⟨2 ≤ m⟩
    by force

  have ?Am ∩ ?Bm = {}
    by blast

moreover

  have Am: ?Am' ∩ ?Am'' = {} finite ?Am' finite ?Am''
    using ⟨?Am' ⊆ ?Sb⟩ ⟨?Am'' ⊆ ?Sa⟩ ⟨?Sa ∩ ?Sb = {}⟩

```

```

    by auto

have finite ?Am finite ?Bm
  by auto

have ?Am  $\cup$  ?Bm = ?S
proof-
  have ?Am  $\subseteq$  ?S
    using  $\langle ?Am' \subseteq ?Sb \rangle \langle ?Am'' \subseteq ?Sa \rangle$ 
    by blast
  then show ?thesis
    by blast
qed

moreover

have card ?Am = m
proof-
  have inj-on  $(\lambda k. 2 * (3::nat) ^ k) \{n-m+1..<n\}$ 
    unfolding inj-on-def
    by auto
  then show ?thesis
    using card-image[of  $\lambda k. 2 * (3::nat) ^ k \{n-m+1..<n\}$ ]
      card-Un-disjoint[of ?Am' ?Am''] Am
    unfolding Setcompr-eq-image
    by (smt Int-insert-right-if1 One-nat-def Suc-eq-plus1 Un-insert-right  $(\{2 * 3 ^ k \mid k. k \in \{n-m+1..<n\}\} \cup \{3 ^ (n-m+1)\}) \cap (\{3 ^ k \mid k. k \in \{1..<n\}\} \cup \{2 * 3 ^ k \mid k. k \in \{1..<n\}\} \cup \{1, (3 ^ n + 9) \text{ div } 2 - 1\} - (\{2 * 3 ^ k \mid k. k \in \{n-m+1..<n\}\} \cup \{3 ^ (n-m+1)\})) = \{\}) \langle 2 \leq m \rangle \langle m \leq n \rangle$ 
      add commute
      add-diff-inverse-nat add-le-cancel-left card.insert card-atLeastLessThan card-empty
      diff-Suc-Suc diff-diff-cancel disjoint-insert(2) finite.emptyI insertCI insert-absorb
      le-trans linorder-not-le one-le-numeral)
  qed

moreover
have  $\sum ?Am = \sum ?Bm$ 
proof-
  have  $(\sum ?Am) = 3 ^ n$ 
  proof-
    have  $\sum ?Am' = (\sum k \in \{n-m+1..<n\}. 2 * 3 ^ k)$ 

```

```

proof–
  have inj-on ( $\lambda k. 2 * (\mathcal{I}::nat) ^k$ )  $\{n-m+1..<n\}$ 
    unfolding inj-on-def
    by auto
  then show ?thesis
    unfolding Setcompr-eq-image
    by (simp add: sum.reindex-cong)
qed
also have  $\dots = 2 * (\sum_{k \in \{n-m+1..<n\}} \mathcal{I}^k)$ 
  by (simp add: sum-distrib-left)
also have  $\dots = \mathcal{I}^n - \mathcal{I}^{(n-m+1)}$ 
  using sum-geom-nat'[of  $\mathcal{I}$   $n-m+1$   $n$ ]  $\langle m \geq 2 \rangle \langle m \leq n \rangle$ 
  by simp
finally
have  $\sum ?Am' = \mathcal{I}^n - \mathcal{I}^{(n-m+1)}$ 
  .

moreover

have  $\sum ?Am'' = \mathcal{I}^{(n-m+1)}$ 
  by simp

moreover

have  $\sum ?Am = \sum ?Am' + \sum ?Am''$ 
  using Am
  by (simp add: sum.union-disjoint)

ultimately

have  $(\sum ?Am) = (\mathcal{I}^n - \mathcal{I}^{(n-m+1)}) + \mathcal{I}^{(n-m+1)}$ 
  by simp
also have  $\dots = \mathcal{I}^n$ 
proof–
  have  $(\mathcal{I}::nat) ^{(n-m+1)} \leq \mathcal{I}^n$ 
    using  $\langle m \leq n \rangle \langle 2 \leq m \rangle$ 
    by (metis Nat.le-diff-conv2 add commute add-leD2 diff-diff-cancel
diff-le-self one-le-numeral power-increasing)
  then show ?thesis
    by simp

```

```

qed
finally show ?thesis
.
qed

moreover

have  $\sum ?Bm = 3^n$ 
proof-
  have  $\sum ?S = 2 * 3^n$ 
  proof-
    have  $\sum ?Sa = (\sum_{k \in \{1..<n\}} 3^k)$ 
    proof-
      have inj-on (( $\wedge$ ) ( $3::nat$ )) { $1..<n$ }
      unfolding inj-on-def
      by auto
      then show ?thesis
      unfolding Setcompr-eq-image
      by (simp add: sum.reindex-cong)
    qed
  qed

  have  $\sum ?Sa = (3^n - 1) \text{ div } 2 - 1$ 
  proof-
    have inj-on ( $\lambda k. (3::nat) ^ k$ ) { $1..<n$ }
    unfolding inj-on-def
    by auto
    then have  $\sum ?Sa = (\sum_{k \in \{1..<n\}} 3^k)$ 
    unfolding Setcompr-eq-image
    by (simp add: sum.reindex-cong)
    then show ?thesis
    using sum-geom-nat'[of  $3\ 1\ n$ ]  $\langle n \geq 3 \rangle$ 
    by simp
  qed

  moreover

  have  $\sum ?Sb = 2 * ((3^n - 1) \text{ div } 2 - 1)$ 
  proof-
    have inj-on ( $\lambda k. 2 * (3::nat) ^ k$ ) { $1..<n$ }
    unfolding inj-on-def

```



```

    by auto
  then have  $\sum ?Sb = (\sum k \in \{1..<n\}. 2 * 3 ^ k)$ 
    unfolding Setcompr-eq-image
    by (simp add: sum.reindex-cong)
  also have  $\dots = 2 * (\sum k \in \{1..<n\}. 3 ^ k)$ 
    by (simp add: sum-distrib-left)
  also have  $\dots = 2 * (\sum ?Sa)$ 
  proof-
    have inj-on  $(\lambda k. (3::nat) ^ k) \{1..<n\}$ 
      unfolding inj-on-def
      by auto
    then show ?thesis
      unfolding Setcompr-eq-image
      by (simp add: sum.reindex-cong)
  qed
  finally
  show ?thesis
    using  $\langle \sum ?Sa = (3^n - 1) \text{ div } 2 - 1 \rangle$ 
    by simp
  qed

```

moreover

```

have  $\sum ?Sc = (3 ^ n + 9) \text{ div } 2$ 
  by auto

```

moreover

```

have  $\sum ?S = \sum ?Sa + \sum ?Sb + \sum ?Sc$ 
  using  $\langle ?Sa \cap ?Sb = \{\} \rangle \langle (?Sa \cup ?Sb) \cap ?Sc = \{\} \rangle$ 
  using  $\langle \text{finite } ?Sa \rangle \langle \text{finite } ?Sb \rangle \langle \text{finite } ?Sc \rangle \langle \text{finite } (?Sa \cup ?Sb) \rangle$ 
  using sum.union-disjoint
  by (metis (no-types, lifting))

```

moreover

```

  have  $((3::nat) ^ n - 1) \text{ div } 2 - 1 + 2 * ((3^n - 1) \text{ div } 2 - 1) + (3 ^ n + 9) \text{ div } 2 = 2 * 3 ^ n$  (is ?lhs =  $2 * 3 ^ n$ )
  proof-
    have  $((3::nat) ^ n - 1) \text{ div } 2 - 1 = (3^n - 3) \text{ div } 2$ 

```

```

    by simp
  then have ?lhs = 3*((3^n - 3) div 2) + (3 ^ n + 9) div 2
    by simp
  also have ... = ((3*3^n - 9) + (3^n + 9)) div 2
    by (simp add: div-mult-swap)
  also have ... = 2*3^n
  proof-
    have 9 ≤ (3::nat) * 3 ^ n
      using ⟨n ≥ 3⟩
      by (smt Suc-1 ⟨3 ^ n - 1⟩ div 2 - 1 = (3 ^ n - 3) div 2⟩ cal-
        culation diff-add-inverse2 diff-diff-cancel diff-is-0-eq dvd-mult-div-cancel even-add
        even-power le-add1 le-add-same-cancel2 le-antisym le-trans linear mult-Suc numeral-3-eq-3
        odd-two-times-div-two-succ plus-1-eq-Suc power-mult self-le-ge2-pow)
    then have ((3::nat)*3^n - 9) + (3^n + 9) = 4*3^n
      by simp
    then show ?thesis
      by simp
  qed
  finally
  show ?thesis
  .
  qed

  ultimately
  show ?thesis
    by simp
  qed
  also have  $\sum ?S = \sum ?Am + \sum ?Bm$ 
    using ⟨?Am ∪ ?Bm = ?S⟩ ⟨?Am ∩ ?Bm = {}⟩ ⟨finite ?Am⟩ ⟨finite ?Bm⟩
    using sum.union-disjoint[of ?Am ?Bm id]
    by simp
  then show ?thesis
    using ⟨ $\sum ?Am = 3^n$ ⟩
    by (metis (no-types, lifting) add-left-cancel calculation mult-2)
  qed

  ultimately

  show ?thesis
    by simp

```

```

qed

ultimately

show  $\exists S1\ S2. S1 \cap S2 = \{\} \wedge S1 \cup S2 = ?S \wedge \text{card } S1 = m \wedge \sum S1 = \sum S2$ 
by blast
qed
qed

end

```

5.2.2 IMO 2018 SL - C2

```

theory IMO-2018-SL-C2-sol
imports Complex-Main
begin

locale dim =
  fixes files :: int
  fixes ranks :: int
  assumes pos: files > 0  $\wedge$  ranks > 0
  assumes div4: files mod 4 = 0  $\wedge$  ranks mod 4 = 0
begin

type-synonym square = int  $\times$  int

definition squares :: square set where
  squares = {0.. $\text{files}$ }  $\times$  {0.. $\text{ranks}$ }

datatype piece = Queen | Knight

type-synonym board = square  $\Rightarrow$  piece option

definition empty-board :: board where
  empty-board = ( $\lambda$  square. None)

fun attacks-knight :: square  $\Rightarrow$  board  $\Rightarrow$  bool where
  attacks-knight (file, rank) board  $\longleftrightarrow$ 
    ( $\exists$  file' rank'. (file', rank')  $\in$  squares  $\wedge$  board (file', rank') = Some Knight  $\wedge$ 

```

$$((\text{abs } (\text{file} - \text{file}') = 1 \wedge \text{abs } (\text{rank} - \text{rank}') = 2) \vee \\ (\text{abs } (\text{file} - \text{file}') = 2 \wedge \text{abs } (\text{rank} - \text{rank}') = 1)))$$

definition *valid-horst-move'* :: *square* \Rightarrow *board* \Rightarrow *board* \Rightarrow *bool* **where**
valid-horst-move' *square board board'* \longleftrightarrow
square \in *squares* \wedge *board square* = *None* \wedge
 \neg *attacks-knight square board* \wedge
board' = *board* (*square* := *Some Knight*)

definition *valid-horst-move* :: *board* \Rightarrow *board* \Rightarrow *bool* **where**
valid-horst-move board board' \longleftrightarrow
 $(\exists \text{ square. } \text{valid-horst-move}' \text{ square board board'})$

definition *valid-queenie-move* :: *board* \Rightarrow *board* \Rightarrow *bool* **where**
valid-queenie-move board board' \longleftrightarrow
 $(\exists \text{ square} \in \text{squares. board square} = \text{None} \wedge$
board' = *board* (*square* := *Some Queen*))

type-synonym *strategy* = *board* \Rightarrow *board* \Rightarrow *bool*

inductive *valid-game* :: *strategy* \Rightarrow *strategy* \Rightarrow *nat* \Rightarrow *board* \Rightarrow *bool* **where**
valid-game horst-strategy queenie-strategy 0 empty-board
 $| \llbracket \text{valid-game horst-strategy queenie-strategy } k \text{ board};$
valid-horst-move board board'; horst-strategy board board';
valid-queenie-move board' board''; queenie-strategy board' board'' \rrbracket \Longrightarrow \text{valid-game}
horst-strategy queenie-strategy (k + 1) board''

definition *valid-queenie-strategy* :: *strategy* \Rightarrow *bool* **where**
valid-queenie-strategy queenie-strategy \longleftrightarrow
 $(\forall \text{ horst-strategy board board' } k.$
valid-game horst-strategy queenie-strategy k board \wedge
valid-horst-move board board' \wedge horst-strategy board board' \wedge
 $(\exists \text{ square} \in \text{squares. board' square} = \text{None}) \longrightarrow$
 $(\exists \text{ board''. valid-queenie-move board' board''} \wedge \text{queenie-strategy board'}$
board''))

squares

lemma *squares-card* [*simp*]:
shows *card squares* = *files* * *ranks*
using *pos*

unfolding *squares-def*
by *auto*

lemma *squares-finite* [*simp*]:
shows *finite squares*
using *pos*
unfolding *squares-def*
by *auto*

free-squares

definition *free-squares* :: *board* \Rightarrow *square set* **where**
free-squares board = {*square* \in *squares*. *board square* = *None*}

lemma *free-squares-finite* [*simp*]:
shows *finite (free-squares board)*
proof (*rule finite-subset*)
show *free-squares board* \subseteq *squares*
by (*simp add: free-squares-def*)
qed *simp*

lemma *valid-game-free-squares-card-even*:
assumes *valid-game horst-strategy queenie-strategy k board*
shows *card (free-squares board) mod 2 = 0*
using *assms*
proof (*induction horst-strategy queenie-strategy k board rule: valid-game.induct*)
case (*1 horst-strategy queenie-strategy*)
show ?*case*
proof–
have *card (free-squares empty-board) = files * ranks*
by (*simp add: empty-board-def free-squares-def*)
then show ?*thesis*
using *div4*
by *presburger*
qed
next
case (*2 horst-strategy queenie-strategy K board board' board''*)
then obtain *square square'* **where**
square \in *squares board square = None board' = board (square := Some Knight)*
square' \in squares board' square' = None board'' = board' (square' := Some Queen)

```

    unfolding valid-horst-move-def valid-horst-move'-def valid-queenie-move-def
    by auto
  then have free-squares board = free-squares board''  $\cup$  {square, square'}
    square  $\notin$  free-squares board'' square'  $\notin$  free-squares board''
    unfolding free-squares-def
    by (auto split: if-split-asm)
  moreover
  have square  $\neq$  square'
    using  $\langle \text{board}' = \text{board}(\text{square} \mapsto \text{Knight}) \rangle \langle \text{board}' \text{ square}' = \text{None} \rangle$ 
    by auto
  ultimately
  have card (free-squares board) = card (free-squares board'') + 2
    using card-Un-disjoint[of free-squares board'' {square, square'}]
    by auto
  then show ?case
    using  $\langle \text{card} (\text{free-squares board}) \bmod 2 = 0 \rangle$ 
    by simp
qed

```

black squares

```

fun black :: square  $\Rightarrow$  bool where
  black (file, rank)  $\longleftrightarrow$  (file + rank) mod 2 = 0

```

```

definition black-squares :: square set where
  black-squares = {square  $\in$  squares. black square}

```

```

lemma black-squares-finite [simp]:
  shows finite black-squares
  using pos
  unfolding black-squares-def
  by auto

```

```

lemma black-squares-card:
  card black-squares = (files * ranks) div 2

```

```

proof-
  let ?black-squares = {square  $\in$  squares. black square}
  let ?white-squares = {square  $\in$  squares.  $\neg$  black square}
  have squares = ?black-squares  $\cup$  ?white-squares
    by blast
  moreover

```

```

have ?black-squares  $\cap$  ?white-squares = {}
  by blast
moreover
have card ?black-squares = card ?white-squares
proof-
  let ?f =  $\lambda$  (a::int, b::int). if a mod 2 = 0 then (a, b + 1) else (a, b - 1)
  have bij-betw ?f ?black-squares ?white-squares
    unfolding bij-betw-def
  proof
    show inj-on ?f ?black-squares
      unfolding inj-on-def
      by auto
  next
    show ?f ' ?black-squares = ?white-squares
  proof
    show ?f ' ?black-squares  $\subseteq$  ?white-squares
      using div4
      by (auto simp add: squares-def split: if-split-asm) presburger+
  next
    show ?white-squares  $\subseteq$  ?f ' ?black-squares
  proof
    fix wsq
    assume wsq  $\in$  ?white-squares
    let ?invf =  $\lambda$  (a, b). if a mod 2 = 0 then (a, b - 1) else (a, b + 1)
    have ?f (?invf wsq) = wsq
      by (cases wsq, auto)
    moreover
    have ?invf wsq  $\in$  ?black-squares
      using (wsq  $\in$  ?white-squares) div4
      by (cases wsq, auto simp add: squares-def) presburger+
    ultimately
    show wsq  $\in$  ?f ' ?black-squares
      by force
  qed
qed
qed
qed
then show ?thesis
  using bij-betw-same-card by blast
qed
ultimately

```

```

have 2 * card ?black-squares = card squares
  by (metis (no-types, lifting) card.infinite card-Un-disjoint finite-Un mult-2
mult-eq-0-iff)
then have 2 * card ?black-squares = files * ranks
  by auto
then show ?thesis
  unfolding black-squares-def
  by simp
qed

```

free black squares

```

definition free-black-squares :: board  $\Rightarrow$  square set where
  free-black-squares board = {square  $\in$  squares. black square  $\wedge$  board square =
None}

```

lemma free-black-squares-add-piece:

```

  shows card (free-black-squares board)  $\leq$  card (free-black-squares (board (square
:= Some piece))) + 1

```

proof –

```

  let ?board' = board (square := Some piece)
  have free-black-squares board = free-black-squares ?board'  $\vee$ 
    free-black-squares board = free-black-squares ?board'  $\cup$  {square}
  unfolding free-black-squares-def Let-def
  by auto
  then show ?thesis
    by (metis One-nat-def add.right-neutral add-Suc-right card.infinite card-Un-le
card-empty card-insert-if finite-Un finite-insert insert-absorb insert-not-empty le-add1
trans-le-add2)
qed

```

lemma free-black-squares-valid-horst-move:

```

  assumes valid-horst-move board board'
  shows card (free-black-squares board)  $\leq$  card (free-black-squares board') + 1
  using assms
  using free-black-squares-add-piece
  unfolding valid-horst-move-def valid-horst-move'-def free-black-squares-def
  by auto

```

lemma free-black-squares-valid-queenie-move:

```

  assumes valid-queenie-move board board'

```



```

shows card (free-black-squares board)  $\leq$  card (free-black-squares board') + 1
using assms
using free-black-squares-add-piece
unfolding valid-queenie-move-def free-black-squares-def
by auto

```

knight

```

definition knight :: board  $\Rightarrow$  square set where
  knight board = {square  $\in$  squares. board square = Some Knight}

```

```

lemma knight-finite [simp]:
  shows finite (knight board)
  by (rule finite-subset[of - squares], simp-all add: knight-def)

```

```

lemma knight-card-horst-move [simp]:
  assumes valid-horst-move board board'
  shows card (knight board') = card (knight board) + 1

```

proof–

```

  obtain square where square  $\in$  squares board square = None board' square =
  Some Knight
  board' = board (square := Some Knight)
  using assms
  unfolding valid-horst-move-def valid-horst-move'-def
  by auto
  then have knight board' = knight board  $\cup$  {square}
  unfolding knight-def
  by auto
  then show ?thesis
  using ⟨board square = None⟩
  unfolding knight-def
  by auto

```

qed

```

lemma knight-card-queenie-move [simp]:
  assumes valid-queenie-move board board'
  shows card (knight board') = card (knight board)

```

proof–

```

  have knight board' = knight board
  using assms
  unfolding valid-queenie-move-def knight-def

```

```

    by force
  then show ?thesis
    by simp
qed

```

```

lemma valid-game-knights-card [simp]:
  assumes valid-game horst-strategy queenie-strategy k board
  shows card (knights board) = k
  using assms
proof (induction horst-strategy queenie-strategy k board rule: valid-game.induct)
  case (1 horst-strategy queenie-strategy)
  show ?case
    by (simp add: empty-board-def knights-def)
next
  case (2 horst-strategy queenie-strategy K board board' board'')
  then show ?case
    by auto
qed

```

Cycles

```

fun cycle-opposite :: square  $\Rightarrow$  square where
  cycle-opposite (file, rank) = (4 * (file div 4) + (3 - file mod 4), 4 * (rank div 4) + (3 - rank mod 4))

```

```

lemma cycle-opposite-cycle-opposite [simp]:
  shows cycle-opposite (cycle-opposite square) = square
  by (cases square) auto

```

```

lemma cycle-opposite-different [simp]:
  shows cycle-opposite square  $\neq$  square
  by (cases square, simp, presburger)

```

```

lemma cycle-opposite-squares [simp]:
  shows cycle-opposite square  $\in$  squares  $\longleftrightarrow$  square  $\in$  squares
  using pos div4
  by (cases square) (simp add: squares-def, safe, presburger+)

```

```

fun cycle4 :: square  $\Rightarrow$  int where
  cycle4 (x, y) =

```

```

(if  $x = 0$  then  $y$ 
 else if  $x = 1$  then  $(y + 2) \bmod 4$ 
 else if  $x = 2$  then  $(5 - y) \bmod 4$ 
 else  $3 - y$ )

```

lemma *cycle-lt-4*:

```

assumes  $0 \leq x \ x < 4 \ 0 \leq y \ y < 4$ 
shows  $0 \leq \text{cycle4 } (x, y) \wedge \text{cycle4 } (x, y) < 4$ 
using assms
by auto

```

lemma *cycle0*:

```

assumes  $0 \leq x \ x < 4 \ 0 \leq y \ y < 4$ 
shows  $\text{cycle4 } (x, y) = 0 \longleftrightarrow (x, y) \in \text{set } [(0, 0), (2, 1), (1, 2), (3, 3)]$ 
using assms
by auto presburger+

```

lemma *cycle1*:

```

assumes  $0 \leq x \ x < 4 \ 0 \leq y \ y < 4$ 
shows  $\text{cycle4 } (x, y) = 1 \longleftrightarrow (x, y) \in \text{set } [(0, 1), (1, 3), (3, 2), (2, 0)]$ 
using assms
by auto presburger+

```

lemma *cycle2*:

```

assumes  $0 \leq x \ x < 4 \ 0 \leq y \ y < 4$ 
shows  $\text{cycle4 } (x, y) = 2 \longleftrightarrow (x, y) \in \text{set } [(0, 2), (2, 3), (1, 0), (3, 1)]$ 
using assms
by auto presburger+

```

lemma *cycle3*:

```

assumes  $0 \leq x \ x < 4 \ 0 \leq y \ y < 4$ 
shows  $\text{cycle4 } (x, y) = 3 \longleftrightarrow (x, y) \in \text{set } [(0, 3), (1, 1), (2, 2), (3, 0)]$ 
using assms
by auto presburger+

```

fun *cycle* :: *square* \Rightarrow *int* \times *int* \times *int* **where**

```

cycle  $(x, y) = (x \text{ div } 4, y \text{ div } 4, \text{cycle4 } (x \text{ mod } 4, y \text{ mod } 4))$ 

```

lemma *cycles-card*:

```

shows  $\text{card } (\text{cycle } ^\text{'squares}) = (\text{files} * \text{ranks}) \text{ div } 4$ 

```

proof–

have $\text{cycle} \text{ 'squares} = \{(x, y, z). x \in \{0..<\text{files div } 4\} \wedge y \in \{0..<\text{ranks div } 4\} \wedge z \in \{0..<4\}\}$

proof *safe*

fix $f \ r \ x \ y \ z$

assume $(f, r) \in \text{squares} \ (x, y, z) = \text{cycle} \ (f, r)$

then have $0 \leq f \wedge f < \text{files} \ 0 \leq r \wedge r < \text{ranks}$

by $(\text{auto simp add: squares-def})$

then have $0 \leq f \text{ div } 4 \wedge f \text{ div } 4 < \text{files div } 4 \ 0 \leq r \text{ div } 4 \wedge r \text{ div } 4 < \text{ranks div } 4$

using *div4*

by *presburger+*

then show $x \in \{0..<\text{files div } 4\} \ y \in \{0..<\text{ranks div } 4\}$

using $\langle (x, y, z) = \text{cycle} \ (f, r) \rangle$

by *auto*

show $z \in \{0..<4\}$

using $\text{cycle-lt-4}[\text{rule-format, of } f \text{ mod } 4 \ r \text{ mod } 4]$

using $\langle (x, y, z) = \text{cycle} \ (f, r) \rangle$

by *simp*

next

fix $x \ y \ z :: \text{int}$

assume $*: x \in \{0..<\text{files div } 4\} \ y \in \{0..<\text{ranks div } 4\} \ z \in \{0..<4\}$

let $?f = 4 * x$ **and** $?r = 4 * y + z$

have $(?f, ?r) \in \text{squares} \ \text{cycle} \ (?f, ?r) = (x, y, z)$

using $*$

by $(\text{auto simp add: squares-def})$

then have $\exists \text{ square} \in \text{squares}. \text{cycle} \ \text{square} = (x, y, z)$

by *blast*

then show $(x, y, z) \in \text{cycle} \text{ 'squares}$

by (metis imageI)

qed

also have $\dots = \{0..<\text{files div } 4\} \times \{0..<\text{ranks div } 4\} \times \{0..<4\}$

by *auto*

finally

have $\text{card} \ (\text{cycle} \text{ 'squares}) = (\text{files div } 4) * (\text{ranks div } 4) * 4$

using *pos*

by *simp*

also have $\dots = (\text{files} * \text{ranks}) \text{ div } 4$

using *div4*

by *auto*

finally show ?thesis

qed

lemma *cycle4-exhausted*:

assumes $0 \leq f1$ $f1 < 4$ $0 \leq r1$ $r1 < 4$

assumes $0 \leq f2$ $f2 < 4$ $0 \leq r2$ $r2 < 4$

assumes $(f1, r1) \neq (f2, r2)$

$abs\ (f1 - f2) \neq 1 \vee abs\ (r1 - r2) \neq 2$

$abs\ (f1 - f2) \neq 2 \vee abs\ (r1 - r2) \neq 1$

$(f2, r2) \neq (3 - f1, 3 - r1)$

shows $cycle4\ (f1, r1) \neq cycle4\ (f2, r2)$

using *assms cycle-4* [rule-format, of *f1 r1*]

by (*smt cycle0 cycle1 cycle2 cycle3 list.set-intros(1) list.set-intros(2)*)

lemma *cycle-exhausted*:

assumes $\forall sq \in squares. board\ sq = Some\ Knight \longrightarrow \neg attacks-knight\ sq\ board$

$\forall sq \in squares. board\ sq = Some\ Knight \longrightarrow board\ (cycle-opposite\ sq) =$

Some Queen

$sq1 \neq sq2\ sq1 \in squares\ sq2 \in squares\ board\ sq1 = Some\ Knight\ board$

$sq2 = Some\ Knight$

shows $cycle\ sq1 \neq cycle\ sq2$

proof *safe*

assume $cycle\ sq1 = cycle\ sq2$

obtain *f1 r1* **where** $sq1: sq1 = (f1, r1)$

by (*cases sq1*)

obtain *f2 r2* **where** $sq2: sq2 = (f2, r2)$

by (*cases sq2*)

have **: $f1\ div\ 4 = f2\ div\ 4\ r1\ div\ 4 = r2\ div\ 4$

$cycle4\ (f1\ mod\ 4, r1\ mod\ 4) = cycle4\ (f2\ mod\ 4, r2\ mod\ 4)$

using (*cycle sq1 = cycle sq2*) *sq1 sq2*

by *simp-all*

have $\neg attacks-knight\ (f1, r1)\ board\ (f2, r2) \neq cycle-opposite\ (f1, r1)$

using *assms(1)* [rule-format, of *(f1, r1)*]

using *assms(2)* [rule-format, of *(f1, r1)*]

using *assms(4-7)* *sq1 sq2*

by *auto*

```

have  $f2 \neq 4 * (f1 \text{ div } 4) + (3 - f1 \text{ mod } 4) \vee r2 \neq 4 * (r1 \text{ div } 4) + (3 - r1 \text{ mod } 4)$ 
using  $\langle f2, r2 \rangle \neq \text{cycle-opposite } (f1, r1)$ 
by auto

then have  $f2 \text{ mod } 4 \neq 3 - f1 \text{ mod } 4 \vee r2 \text{ mod } 4 \neq 3 - r1 \text{ mod } 4$ 
using  $**(1-2)$ 
by safe presburger+

then have 1:  $(f2 \text{ mod } 4, r2 \text{ mod } 4) \neq (3 - f1 \text{ mod } 4, 3 - r1 \text{ mod } 4)$ 
by simp

have  $(|f1 - f2| = 1 \longrightarrow |r1 - r2| \neq 2) \wedge (|f1 - f2| = 2 \longrightarrow |r1 - r2| \neq 1)$ 
using  $\langle \neg \text{attacks-knight } (f1, r1) \text{ board} \rangle$ 
using assms attacks-knight.simps sq1 sq2
by blast

then have 2:  $|f1 \text{ mod } 4 - f2 \text{ mod } 4| \neq 1 \vee |r1 \text{ mod } 4 - r2 \text{ mod } 4| \neq 2$ 
 $|f1 \text{ mod } 4 - f2 \text{ mod } 4| \neq 2 \vee |r1 \text{ mod } 4 - r2 \text{ mod } 4| \neq 1$ 
using  $**(1-2)$ 
by  $(\text{smt mult-div-mod-eq})+$ 

have  $(f1 \text{ mod } 4, r1 \text{ mod } 4) = (f2 \text{ mod } 4, r2 \text{ mod } 4)$ 
using  $**(3) \text{ cycle4-exhausted}[OF \text{ - - - - - } 2 \ 1]$ 
using pos-mod-conj zero-less-numeral
by blast

then have  $f1 = f2 \wedge r1 = r2$ 
using  $**(1-2)$ 
by  $(\text{metis mult-div-mod-eq prod.inject})+$ 

then show False
using  $sq1 \ sq2 \ \langle sq1 \neq sq2 \rangle$ 
by simp
qed

```

guaranteed game lengths

definition *guaranteed-game-lengths* :: nat set **where**

guaranteed-game-lengths = $\{K. \exists \text{ horst-strategy. } \forall \text{ queenie-strategy. } \text{valid-queenie-strategy} \text{ queenie-strategy} \longrightarrow (\exists \text{ board. } \text{valid-game horst-strategy queenie-strategy } K \text{ board})\}$

```

lemma guaranteed-game-lengths-geq:
  shows  $\text{nat } ((\text{files} * \text{ranks}) \text{ div } 4) \in \text{guaranteed-game-lengths}$ 
  unfolding guaranteed-game-lengths-def
proof safe
  let  $?l = \text{nat } ((\text{files} * \text{ranks}) \text{ div } 4)$ 
  let  $?horst\text{-}strategy = \lambda \text{ board board}' :: \text{board}. (\exists \text{ square}. \text{black square} \wedge \text{valid-horst-move}'$ 
square board board')
  show  $\exists \text{ horst-strategy}. \forall \text{ queenie-strategy}. \text{valid-queenie-strategy queenie-strategy}$ 
 $\longrightarrow (\exists \text{ board}. \text{valid-game horst-strategy queenie-strategy ?l board})$ 
  proof (rule-tac x=?horst-strategy in exI, safe)
    fix queenie-strategy
    assume valid-queenie-strategy queenie-strategy

    have  $1: \forall k \text{ board}. \text{valid-game } ?horst\text{-}strategy \text{ queenie-strategy } k \text{ board} \longrightarrow (\forall$ 
square  $\in \text{squares}. \text{board square} = \text{Some Knight} \longrightarrow \text{black square})$  (is  $\forall k. ?P k$ )
    proof safe
      fix  $k \text{ board } f \text{ r}$ 
      assume valid-game ?horst-strategy queenie-strategy k board
       $(f, r) \in \text{squares board } (f, r) = \text{Some Knight}$ 
      then show black (f, r)
      proof (induction ?horst-strategy queenie-strategy k board rule: valid-game.induct)
        case (1 queenie-strategy)
          then show ?case
          by (simp add: empty-board-def)
        next
          case (2 queenie-strategy K board board' board'')
          then show ?case
          by (smt map-upd-Some-unfold piece.simps(1) valid-horst-move'-def
valid-queenie-move-def)
        qed
      qed

    have  $\forall k \leq (\text{files} * \text{ranks}) \text{ div } 4. \exists \text{ board}. \text{valid-game } ?horst\text{-}strategy \text{ queenie-strategy}$ 
k board
    proof safe
      fix  $k :: \text{nat}$ 
      assume  $k \leq (\text{files} * \text{ranks}) \text{ div } 4$ 
      then show  $\exists \text{ board}. \text{valid-game } ?horst\text{-}strategy \text{ queenie-strategy } k \text{ board}$ 
      proof (induction k)

```

```

    case 0
    then show ?case
      by (rule-tac x=empty-board in exI, simp add: valid-game.intros)
  next
  case (Suc k)
    then obtain board where valid-game ?horst-strategy queenie-strategy k
board
      by auto
    then have *: (files * ranks) div 2 - 2 * k ≤ card (free-black-squares board)
      using ⟨Suc k ≤ (files * ranks) div 4⟩
  proof (induction ?horst-strategy queenie-strategy k board rule: valid-game.induct)
    case 1
    then show ?case
      using black-squares-card
      by (simp add: empty-board-def black-squares-def free-black-squares-def)
  next
    case (2 queenie-strategy k board board' board'')
    then have (files * ranks) div 2 - 2 * k ≤ card (free-black-squares board)
      by auto
    also have ... ≤ card (free-black-squares board') + 1
      using 2
      using free-black-squares-valid-horst-move[of board board']
      by simp
    also have ... ≤ card (free-black-squares board'') + 2
      using 2
      using free-black-squares-valid-queenie-move[of board' board'']
      by simp
    finally show ?case
      using ⟨Suc (k + 1) ≤ (files * ranks) div 4⟩
      by (simp add: le-diff-conv)
  qed
  then have card (free-black-squares board) > 0
    using ⟨Suc k ≤ (files * ranks) div 4⟩
    by auto
  then obtain square where square ∈ free-black-squares board
  by (metis Collect-empty-eq Collect-mem-eq card.infinite card-0-eq not-less0)

  have ¬ attacks-knight square board
  proof (rule ccontr)
    obtain x y where square = (x, y)

```



```

    by (cases square)
    assume  $\neg$  ?thesis
    then obtain  $x' y'$  where  $(x', y') \in \text{squares board}$   $(x', y') = \text{Some Knight}$ 
 $|x - x'| = 1 \wedge |y - y'| = 2 \vee |x - x'| = 2 \wedge |y - y'| = 1$ 
    using  $\langle \text{square} = (x, y) \rangle$ 
    by auto
    then have black  $(x', y')$ 
    using  $1[\text{rule-format}, OF \langle \text{valid-game ?horst-strategy queenie-strategy k}$ 
board $\rangle]$ 
    by auto

    have black  $(x, y)$ 
    using  $\langle \text{square} \in \text{free-black-squares board} \rangle \langle \text{square} = (x, y) \rangle$ 
    by (simp add: free-black-squares-def)

    show False
    using  $\langle \text{black}(x, y) \rangle \langle \text{black}(x', y') \rangle \langle |x - x'| = 1 \wedge |y - y'| = 2 \vee |x -$ 
 $x'| = 2 \wedge |y - y'| = 1 \rangle$ 
    unfolding black.simps
    by presburger
qed

let ?board1 = board (square := Some Knight)
have valid-horst-move board ?board1
    using  $\langle \text{square} \in \text{free-black-squares board} \rangle \langle \neg \text{attacks-knight square board} \rangle$ 
    unfolding valid-horst-move-def valid-horst-move'-def
by (rule-tac  $x=\text{square}$  in exI, cases square, simp add: free-black-squares-def)

moreover

have ?horst-strategy board ?board1
    using  $\langle \text{valid-horst-move board ?board1} \rangle \langle \text{square} \in \text{free-black-squares board} \rangle$ 
    unfolding valid-horst-move-def free-black-squares-def
    by (rule-tac  $x=\text{square}$  in exI, cases square)
    (metis (mono-tags, lifting) map-upd-Some-unfold mem-Collect-eq op-
tion.discI valid-horst-move'-def)

moreover

have  $\exists \text{ square} \in \text{squares. ?board1 square} = \text{None}$ 

```

```

proof–
  have  $\text{card } (\text{free-squares board}) \bmod 2 = 0$ 
    using  $\langle \text{valid-game } ?\text{horst-strategy queenie-strategy } k \text{ board} \rangle$ 
    using  $\text{valid-game-free-squares-card-even}$ 
    by blast
    have  $\text{free-squares board} = \text{free-squares } ?\text{board1} \cup \{\text{square}\} \text{ square} \notin$ 
 $\text{free-squares } ?\text{board1}$ 
    using  $\langle \text{square} \in \text{free-black-squares board} \rangle$ 
    unfolding  $\text{free-black-squares-def free-squares-def}$ 
    by auto
  then have  $\text{card } (\text{free-squares board}) = \text{card } (\text{free-squares } ?\text{board1}) + 1$ 
    by auto
  then have  $\text{card } (\text{free-squares } ?\text{board1}) \bmod 2 = 1$ 
    using  $\langle \text{card } (\text{free-squares board}) \bmod 2 = 0 \rangle$ 
    by presburger
  then have  $\text{free-squares } ?\text{board1} \neq \{\}$ 
    by auto
  then show  $?thesis$ 
    unfolding  $\text{free-squares-def}$ 
    by blast
qed

  then obtain  $\text{board2}$  where  $\text{valid-queenie-move } ?\text{board1 board2 queenie-strategy}$ 
 $?board1 \text{ board2}$ 
    using  $\langle \text{valid-queenie-strategy queenie-strategy} \rangle$ 
    unfolding  $\text{valid-queenie-strategy-def}$ 
    using  $\langle \text{valid-game } ?\text{horst-strategy queenie-strategy } k \text{ board} \rangle \text{ calculation}(1)$ 
 $\text{calculation}(2) \text{ valid-horst-move'-def}$ 
    by blast

  ultimately

  show  $?case$ 
    using  $\langle \text{valid-game } ?\text{horst-strategy queenie-strategy } k \text{ board} \rangle$ 
    by  $(\text{metis } (\text{no-types, lifting}) \text{ Suc-eq-plus1 } \text{valid-game.intros}(2))$ 
qed
qed
then show  $\exists \text{ board. valid-game } ?\text{horst-strategy queenie-strategy } ?l \text{ board}$ 
  using pos
  by simp

```

```

qed
qed

lemma valid-game-not-attacks-knight:
  assumes valid-game horst-strategy queenie-strategy k board
    square  $\in$  squares board square = Some Knight
  shows  $\neg$  attacks-knight square board
  using assms
proof (induction horst-strategy queenie-strategy k board rule: valid-game.induct)
  case (1 horst-strategy queenie-strategy)
  then show ?case
    by (simp add: empty-board-def)
next
  case (2 horst-strategy queenie-strategy K board board' board'')
  have  $\neg$  attacks-knight square board'
  proof (cases board square = Some Knight)
    case True
    then have  $\neg$  attacks-knight square board
      using 2
      by simp
    show ?thesis
  proof (rule ccontr)
    assume  $\neg$  ?thesis
    obtain x y where square = (x, y)
      by (cases square)
    then obtain x' y' where (x', y')  $\in$  squares board' (x', y') = Some Knight
       $|x - x'| = 1 \wedge |y - y'| = 2 \vee |x - x'| = 2 \wedge |y - y'| = 1$ 
      using  $\langle \neg \neg$  attacks-knight square board'  $\rangle$ 
      by auto
    obtain square' where
      square'  $\in$  squares  $\neg$  attacks-knight square' board
      board square' = None board' = board (square' := Some Knight)
      using  $\langle$  valid-horst-move board board'  $\rangle$ 
      unfolding valid-horst-move-def valid-horst-move'-def
      by auto
    have square' = (x', y')
      using  $\langle |x - x'| = 1 \wedge |y - y'| = 2 \vee |x - x'| = 2 \wedge |y - y'| = 1 \rangle$ 
      using  $\langle \neg$  attacks-knight square board  $\rangle$   $\langle$  board' (x', y') = Some Knight  $\rangle$   $\langle$  board'
      = board (square'  $\mapsto$  Knight)  $\rangle$   $\langle$  (x', y')  $\in$  squares  $\rangle$   $\langle$  square = (x, y)  $\rangle$ 
      by (metis (full-types) attacks-knight.simps fun-upd-other)
  end
  end
end

```

```

then have attacks-knight square' board
  using  $\langle \text{square}' \in \text{squares} \rangle \langle |x - x'| = 1 \wedge |y - y'| = 2 \vee |x - x'| = 2 \wedge |y - y'| = 1 \rangle$ 
     $\langle \text{board square} = \text{Some Knight} \rangle \langle \text{square} = (x, y) \rangle$ 
  using  $\langle \text{square} \in \text{squares} \rangle \langle \text{board square} = \text{Some Knight} \rangle$ 
  by (smt attacks-knight.simps)
then show False
  using  $\langle \neg \text{attacks-knight square}' \text{ board} \rangle$ 
  by simp
qed
next
  case False
  have board' square = Some Knight
    using  $\langle \text{square} \in \text{squares} \rangle \langle \text{board'' square} = \text{Some Knight} \rangle \langle \text{valid-queenie-move board' board''} \rangle$ 
    by (metis map-upd-Some-unfold piece.distinct(1) valid-queenie-move-def)

obtain square' where *: square' ∈ squares
  board square' = None  $\neg \text{attacks-knight square}' \text{ board}$ 
  board' = board(square' ↦ Knight)
  using  $\langle \text{valid-horst-move board board'} \rangle$ 
  unfolding valid-horst-move-def valid-horst-move'-def
  by blast
then have square = square'
  using  $\langle \text{board square} \neq \text{Some Knight} \rangle$ 
  using  $\langle \text{board' square} = \text{Some Knight} \rangle$ 
  by (metis fun-upd-apply)
then have  $\neg \text{attacks-knight square board}$ 
  using  $\langle \neg \text{attacks-knight square}' \text{ board} \rangle$ 
  by simp
then show ?thesis
  by (cases square) (simp add: *(4) ⟨square = square'⟩)
qed
then show ?case
  using  $\langle \text{valid-queenie-move board' board''} \rangle$ 
  by (smt attacks-knight.elims(2) attacks-knight.elims(3) fun-upd-apply option.inject piece.simps(1) prod.simps(1) valid-queenie-move-def)
qed

```

lemma *guaranteed-game-lengths-leq*:

```

shows  $\forall k \in \text{guaranteed-game-lengths}. k \leq (\text{files} * \text{ranks}) \text{ div } 4$ 
proof safe
  fix  $k$ 
  assume  $k \in \text{guaranteed-game-lengths}$ 
  then obtain horst-strategy where
    *:  $\forall \text{ queenie-strategy}. \text{valid-queenie-strategy } \text{queenie-strategy} \longrightarrow$ 
       $(\exists \text{ board}. \text{valid-game } \text{horst-strategy } \text{queenie-strategy } k \text{ board})$ 
  unfolding guaranteed-game-lengths-def
  by auto
  show  $k \leq (\text{files} * \text{ranks}) \text{ div } 4$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then have  $k > (\text{files} * \text{ranks}) \text{ div } 4$ 
    by simp

    let  $?queenie\text{-}strategy = \lambda \text{ board } \text{board}'. (\exists \text{ square} \in \text{squares}. \text{board } \text{square}$ 
     $= \text{Some Knight} \wedge \text{board } (\text{cycle-opposite } \text{square}) = \text{None} \wedge \text{board}' (\text{cycle-opposite}$ 
     $\text{square}) = \text{Some Queen})$ 

    have 1:  $\forall k \text{ horst-strategy } \text{board}. \text{valid-game } \text{horst-strategy } ?queenie\text{-}strategy \text{ } k$ 
     $\text{board} \longrightarrow$ 
       $(\forall \text{ square} \in \text{squares}. \text{board } \text{square} = \text{Some Knight} \longleftrightarrow \text{board}$ 
       $(\text{cycle-opposite } \text{square}) = \text{Some Queen})$  (is  $\forall k. ?P \text{ } k$ )
    proof (rule allI, rule allI, rule allI, rule impI, rule ballI)
      fix  $k \text{ horst-strategy } \text{board } \text{square}$ 
      assume  $\text{valid-game } \text{horst-strategy } ?queenie\text{-}strategy \text{ } k \text{ board } \text{square} \in \text{squares}$ 
      then show  $(\text{board } \text{square} = \text{Some Knight}) = (\text{board } (\text{cycle-opposite } \text{square})$ 
       $= \text{Some Queen})$ 
      proof (induction horst-strategy ?queenie-strategy k board arbitrary: square
      rule: valid-game.induct)
        case (1 horst-strategy)
          then show ?case
          by (simp add: empty-board-def)
        next
          case (2 horst-strategy K board board' board'')
          show ?case
          proof safe
            assume  $\text{board}'' \text{ square} = \text{Some Knight}$ 
            show  $\text{board}'' (\text{cycle-opposite } \text{square}) = \text{Some Queen}$ 
            proof (cases board square = Some Knight)

```

```

case True
then have board (cycle-opposite square) = Some Queen
  using 2
  by blast
then have board' (cycle-opposite square) = Some Queen
  using  $\langle \text{valid-horst-move } \text{board } \text{board}' \rangle$ 
  unfolding valid-horst-move-def valid-horst-move'-def
  by (metis fun-upd-apply option.distinct(1))
then show ?thesis
  using  $\langle \text{valid-queenie-move } \text{board}' \text{ board}'' \rangle$ 
  using valid-queenie-move-def
  by auto
next
  case False
from  $\langle \text{valid-queenie-move } \text{board}' \text{ board}'' \rangle \langle ?\text{queenie-strategy } \text{board}' \text{ board}'' \rangle$ 
obtain square' where
  square'  $\in$  squares
  board' square' = Some Knight
  board' (cycle-opposite square') = None
  board'' (cycle-opposite square') = Some Queen
  by auto

have square = square'
proof (rule ccontr)
  assume square  $\neq$  square'
  then have board square' = Some Knight
    using  $\langle \text{board}'' \text{ square} = \text{Some Knight} \rangle \langle \text{board}' \text{ square}' = \text{Some Knight} \rangle$ 
     $\langle \text{valid-horst-move } \text{board } \text{board}' \rangle \langle \text{valid-queenie-move } \text{board}' \text{ board}'' \rangle$ 
    by (smt False map-upd-Some-unfold piece.distinct(1) valid-horst-move'-def
    valid-horst-move-def valid-queenie-move-def)
    then have board (cycle-opposite square') = Some Queen
      using  $\langle \text{square}' \in \text{squares} \rangle$  2
      by simp
    then have board' (cycle-opposite square') = Some Queen
      by (metis  $\langle \text{board}' (\text{cycle-opposite square}') = \text{None} \rangle \langle \text{valid-horst-move}$ 
board board'  $\rangle$  fun-upd-def valid-horst-move'-def valid-horst-move-def)
    then show False
      using  $\langle \text{board}' (\text{cycle-opposite square}') = \text{None} \rangle$ 
      by simp
qed

```

```

    then show ?thesis
      using ⟨board'' (cycle-opposite square) = Some Queen⟩
      by simp
  qed
next
  assume board'' (cycle-opposite square) = Some Queen
  show board'' square = Some Knight
  proof (cases board (cycle-opposite square) = Some Queen)
    case True
    then have board square = Some Knight
      using 2
      by auto
    then have board' square = Some Knight
      using ⟨valid-horst-move board board'⟩
  unfolding valid-horst-move-def valid-horst-move'-def valid-queenie-move-def
    by auto
    then show ?thesis
      using ⟨valid-queenie-move board' board''⟩
      unfolding valid-queenie-move-def
      by auto
  next
    case False
    then have board' (cycle-opposite square) ≠ Some Queen
      using ⟨valid-horst-move board board'⟩
  unfolding valid-horst-move-def valid-horst-move'-def valid-queenie-move-def
    by (meson map-upd-Some-unfold piece.simps(2))
    obtain square' where square' ∈ squares
      board' (cycle-opposite square') = None
      board'' (cycle-opposite square') = Some Queen
      board' square' = Some Knight
    using ⟨?queenie-strategy board' board''⟩
    by auto
  moreover
    obtain square'' where board' square'' = None
      board'' = board' (square'' := Some Queen)
    using ⟨valid-queenie-move board' board''⟩
    unfolding valid-queenie-move-def
    by auto
  ultimately
    have cycle-opposite square' = square''

```

```

    by (auto split: if-split-asm)
  then have cycle-opposite square' = cycle-opposite square
    using ⟨board'' (cycle-opposite square) = Some Queen⟩
    using ⟨board' (cycle-opposite square) ≠ Some Queen⟩
    using ⟨board'' = board' (square'' := Some Queen)⟩
    by (auto split: if-split-asm)
    then have cycle-opposite (cycle-opposite square') = cycle-opposite
(cycle-opposite square)
    by simp
  then have square' = square
    by simp
  then have board' square = Some Knight
    using ⟨board' square' = Some Knight⟩
    by simp
  then show ?thesis
    using ⟨board'' = board' (square'' ↦ Queen)⟩
      ⟨board' (cycle-opposite square') = None⟩
      ⟨cycle-opposite square' = square''⟩ ⟨square' = square⟩
    by auto
qed
qed
qed
qed

have valid-queenie-strategy ?queenie-strategy
  unfolding valid-queenie-strategy-def
proof safe
  fix horst-strategy board board' k f r
  assume valid-game horst-strategy ?queenie-strategy k board
    valid-horst-move board board' horst-strategy board board'
  then obtain square where
    *: square ∈ squares board square = None ∧ attacks-knight square board board'
= board(square ↦ Knight)
    unfolding valid-horst-move-def valid-horst-move'-def
    by auto
  have board (cycle-opposite square) ≠ Some Queen board (cycle-opposite
square) ≠ Some Knight
    using 1[rule-format, OF ⟨valid-game horst-strategy ?queenie-strategy k
board⟩, of square]
    using 1[rule-format, OF ⟨valid-game horst-strategy ?queenie-strategy k

```



```

board⟩, of cycle-opposite square]
  using ⟨square ∈ squares⟩ ⟨board square = None⟩
  by auto
then have board (cycle-opposite square) = None
  by (metis (full-types) option.exhaust-sel piece.exhaust)

let ?board = board' (cycle-opposite square := Some Queen)
have ?queenie-strategy board' ?board
  using * ⟨board (cycle-opposite square) = None⟩ ⟨square ∈ squares⟩
  by (rule-tac x=square in bexI, simp-all)

moreover

obtain f' r' where cycle-opposite square = (f', r')
  by (cases cycle-opposite square)
then have valid-queenie-move board' ?board
  using ⟨board (cycle-opposite square) = None⟩ cycle-opposite-squares[of
square]
  unfolding valid-queenie-move-def
  by (metis *(1) *(4) cycle-opposite-different fun-upd-other)

ultimately
show ∃ board''.
  valid-queenie-move board' board'' ∧
  ?queenie-strategy board' board''
  by blast
qed

then obtain board where **: valid-game horst-strategy ?queenie-strategy k
board
  using *
  by auto

have card (knights board) > (files * ranks) div 4
  using valid-game-knights-card[rule-format, OF **] ⟨k > (files * ranks) div 4⟩
  by auto

have card (cycle ‘ (knights board)) > (files * ranks) div 4
proof-
  have inj-on cycle (knights board)

```

```

unfolding inj-on-def
proof (rule ballI, rule ballI, rule impI)
  fix square1 square2
  assume square1 ∈ knights board square2 ∈ knights board cycle square1 =
cycle square2
  then show square1 = square2
    using 1[rule-format, OF ⟨valid-game horst-strategy ?queenie-strategy k
board⟩]
    using valid-game-not-attacks-knight[rule-format, OF ⟨valid-game horst-strategy
?queenie-strategy k board⟩]
    using cycle-exhausted[of board]
    unfolding knights-def
    by blast
  qed
then show ?thesis
  using ⟨card (knights board) > (files * ranks) div 4⟩
  by (simp add: card-image)
qed

moreover

have cycle ‘ (knights board) ⊆ cycle ‘ squares
  unfolding knights-def
  by auto

moreover

have finite (cycle ‘ squares)
  by simp

ultimately

have card (cycle ‘ squares) > (files * ranks) div 4
  using card-mono
  by (smt zle-int)

then show False
  using cycles-card
  by simp
qed

```

qed

lemma *guaranteed-game-lengths-finite:*

shows *finite guaranteed-game-lengths*

proof (*subst finite-nat-set-iff-bounded-le*)

show $\exists m. \forall n \in \text{guaranteed-game-lengths}. n \leq m$

proof (*rule-tac x=nat ((files*ranks) div 4) in exI*)

show $\forall n \in \text{guaranteed-game-lengths}. n \leq \text{nat} (\text{files} * \text{ranks} \text{ div } 4)$

using *guaranteed-game-lengths-leq pos*

by *auto*

qed

qed

theorem *Max guaranteed-game-lengths = nat ((files * ranks) div 4)*

proof (*rule Max-eqI*)

show $\text{nat} ((\text{files} * \text{ranks}) \text{ div } 4) \in \text{guaranteed-game-lengths}$

using *guaranteed-game-lengths-geq*

by *auto*

next

fix *k*

assume $k \in \text{guaranteed-game-lengths}$

then show $k \leq \text{nat} ((\text{files} * \text{ranks}) \text{ div } 4)$

using *guaranteed-game-lengths-leq*

by *auto*

next

show *finite guaranteed-game-lengths*

using *guaranteed-game-lengths-finite*

by *auto*

qed

end

end

5.2.3 IMO 2018 SL - C3

theory *IMO-2018-SL-C3-sol*

imports *Complex-Main*

begin

General lemmas

lemma *sum-list-int* [*simp*]:

fixes *xs* :: *nat list*

shows $(\sum x \leftarrow xs. \text{int } (f x)) = \text{int } (\sum x \leftarrow xs. f x)$

by (*induction xs, auto*)

lemma *sum-list-comp*:

shows $(\sum x \leftarrow xs. f (g x)) = (\sum x \leftarrow \text{map } g xs. f x)$

by (*induction xs, auto*)

lemma *lt-ceiling-frac*:

assumes $x < \text{ceiling } (a / b) \text{ } b > 0$

shows $x * b < a$

using *assms*

by (*metis (no-types, hide-lams) floor-less-iff floor-uminus-of-int less-ceiling-iff
minus-mult-minus mult-minus-right of-int-0-less-iff of-int-minus of-int-mult pos-less-divide-eq*)

lemma *subset-Max*:

fixes *X* :: *nat set*

assumes *finite X*

shows $X \subseteq \{0..<\text{Max } X + 1\}$

using *assms*

by (*induction X rule: finite.induct*) (*auto simp add: less-Suc-eq-le subsetI*)

lemma *card-Max*:

fixes *X* :: *nat set*

shows $\text{card } X \leq \text{Max } X + 1$

proof (*cases finite X*)

case *True*

then show *?thesis*

using *subset-Max[of X]*

using *subset-eq-atLeast0-lessThan-card* **by** *blast*

next

case *False*

then show *?thesis*

by *simp*

qed

lemma *sum-length-parts*:

```

assumes  $\forall i j. i < j \wedge j < \text{length } ps \longrightarrow \text{set } (\text{filter } (ps ! i) \text{ } xs) \cap \text{set } (\text{filter } (ps$ 
 $! j) \text{ } xs) = \{\}$ 
shows  $\text{sum-list } (\text{map } (\lambda p. \text{length } (\text{filter } p \text{ } xs)) \text{ } ps) \leq \text{length } xs$ 
using assms
proof (induction ps arbitrary: xs)
  case Nil
  then show ?case
    by simp
  next
  case (Cons p ps)
  let  $?xs' = \text{filter } (\lambda x. \neg p \text{ } x) \text{ } xs$ 
  have  $(\sum p \leftarrow ps. \text{length } (\text{filter } p \text{ } xs)) = (\sum p \leftarrow ps. \text{length } (\text{filter } p \text{ } ?xs'))$ 
  proof–
    have *:  $\forall p' \in \text{set } ps. \text{set } (\text{filter } p \text{ } xs) \cap \text{set } (\text{filter } p' \text{ } xs) = \{\}$ 
    using Cons(2)[rule-format, of 0]
    by (metis Suc-less-eq in-set-conv-nth length-Cons list.sel(3) nth-Cons-0 nth-tl
zero-less-Suc)
    have  $\forall p \in \text{set } ps. \text{filter } p \text{ } xs = \text{filter } p \text{ } ?xs'$ 
    proof
      fix  $p'$ 
      assume  $p' \in \text{set } ps$ 
      then have  $\text{set } (\text{filter } p \text{ } xs) \cap \text{set } (\text{filter } p' \text{ } xs) = \{\}$ 
      using *
      by auto
      show  $\text{filter } p' \text{ } xs = \text{filter } p' \text{ } ?xs'$ 
      proof (subst filter-filter, rule filter-cong)
        fix  $x$ 
        assume  $x \in \text{set } xs$ 
        then show  $p' \text{ } x = (\neg p \text{ } x \wedge p' \text{ } x)$ 
        using  $\langle \text{set } (\text{filter } p \text{ } xs) \cap \text{set } (\text{filter } p' \text{ } xs) = \{\} \rangle$ 
        by auto
      qed simp
    qed
    then have  $\forall p \in \text{set } ps. \text{length } (\text{filter } p \text{ } xs) = \text{length } (\text{filter } p \text{ } ?xs')$ 
    by simp
    then show ?thesis
      by (metis (no-types, lifting) map-eq-conv)
    qed
  moreover
  have  $(\sum p \leftarrow ps. \text{length } (\text{filter } p \text{ } (\text{filter } (\lambda x. \neg p \text{ } x) \text{ } xs))) \leq \text{length } (\text{filter } (\lambda x.$ 

```

```

 $\neg p\ x)\ xs)$ 
proof (rule Cons(1), safe)
  fix  $i\ j\ x$ 
  assume  $i < j\ j < \text{length}\ ps\ x \in \text{set}\ (\text{filter}\ (ps\ !\ i)\ ?xs')$   $x \in \text{set}\ (\text{filter}\ (ps\ !\ j)\ ?xs')$ 
  then have False
    using Cons(2)[rule-format, of  $i+1\ j+1$ ]
    by auto
  then show  $x \in \{\}$ 
    by simp
qed

moreover
have  $\text{length}\ (\text{filter}\ p\ xs) + \text{length}\ (\text{filter}\ (\lambda\ x.\ \neg p\ x)\ xs) = \text{length}\ xs$ 
  using sum-length-filter-compl
  by blast

ultimately

show ?case
  by simp
qed

```

```

lemma hd-filter:
  assumes  $\text{filter}\ P\ xs \neq []$ 
  shows  $\exists\ k.\ k < \text{length}\ xs \wedge (\text{filter}\ P\ xs)\ !\ 0 = xs\ !\ k \wedge P\ (xs\ !\ k) \wedge (\forall\ k' < k.\ \neg P\ (xs\ !\ k'))$ 
  using assms
proof (induction xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons x xs)
  show ?case
  proof (cases P x)
    case True
    then show ?thesis
      by auto

```

```

next
  case False
  then obtain k where k < length xs filter P xs ! 0 = xs ! k P (xs ! k) (∀ k' < k.
    ¬ P (xs ! k'))
    using Cons
    by auto
  then show ?thesis
    using False
    by (rule-tac x=k+1 in exI, simp add: nth-Cons')
qed
qed

```

lemma last-filter:

```

  assumes filter P xs ≠ []
  shows ∃ k. k < length xs ∧ (filter P xs) ! (length (filter P xs) - 1) = xs ! k ∧
    P (xs ! k) ∧ (∀ k'. k < k' ∧ k' < length xs → ¬ P (xs ! k'))
proof-
  have filter P (rev xs) ≠ []
    using assms
    by (metis Nil-is-rev-conv rev-filter)
  then obtain k where *: k < length xs filter P (rev xs) ! 0 = rev xs ! k P (rev
    xs ! k) ∀ k' < k. ¬ P (rev xs ! k')
    using hd-filter[of P rev xs]
    by auto
  show ?thesis
  proof (rule-tac x=length xs - (k + 1) in exI, safe)
    show length xs - (k + 1) < length xs
      using *(1)
      by simp
  next
    show filter P xs ! (length (filter P xs) - 1) = xs ! (length xs - (k + 1))
      using *(1) *(2)
      by (metis One-nat-def add.right-neutral add-Suc-right assms length-greater-0-conv
        rev-filter rev-nth)
  next
    show P (xs ! (length xs - (k + 1)))
      using *(1) *(3)
      by (simp add: rev-nth)
  next
    fix k'

```

```

assume  $\text{length } xs - (k + 1) < k' \ k' < \text{length } xs \ P \ (xs \ ! \ k')$ 
then show False
  using  $*(1) \ *(4)[\text{rule-format, of length } xs - (k' + 1)]$ 
  by (smt add.commute add-diff-cancel-right add-diff-cancel-right' add-diff-inverse-nat
add-gr-0 diff-less diff-less-mono2 not-less-eq plus-1-eq-Suc rev-nth zero-less-one)
qed
qed

```

```

lemma filter-tl [simp]:
   $\text{filter } P \ (\text{tl } xs) = (\text{if } P \ (\text{hd } xs) \ \text{then } \text{tl } (\text{filter } P \ xs) \ \text{else } \text{filter } P \ xs)$ 
  by (smt filter.simps(1) filter.simps(2) filter-empty-conv hd-Cons-tl hd-in-set
list.inject list.sel(2))

```

```

lemma filter-dropWhile-not [simp]:
  shows  $\text{filter } P \ (\text{dropWhile } (\lambda x. \neg P \ x) \ xs) = \text{filter } P \ xs$ 
  by (metis (no-types, lifting) filter-False filter-append self-append-conv2 set-takeWhileD
takeWhile-dropWhile-id)

```

```

lemma inside-filter:
  assumes  $i + 1 < \text{length } (\text{filter } P \ xs)$ 
  shows  $\exists \ k1 \ k2. \ k1 < k2 \wedge k2 < \text{length } xs \wedge$ 
     $(\text{filter } P \ xs) \ ! \ i = xs \ ! \ k1 \wedge$ 
     $(\text{filter } P \ xs) \ ! \ (i + 1) = xs \ ! \ k2 \wedge$ 
     $P \ (xs \ ! \ k1) \wedge P \ (xs \ ! \ k2) \wedge$ 
     $(\forall \ k'. \ k1 < k' \wedge k' < k2 \longrightarrow \neg P \ (xs \ ! \ k'))$ 
  using assms
proof (induction i arbitrary: xs)
  case 0
  then obtain k1 where  $k1 < \text{length } xs \ \text{filter } P \ xs \ ! \ 0 = xs \ ! \ k1 \ P \ (xs \ ! \ k1) \ \forall$ 
 $k' < k1. \neg P \ (xs \ ! \ k')$ 
  using hd-filter
  by (metis gr-implies-not-zero length-0-conv)
  let ?xs = drop (k1 + 1) xs
  have  $\text{filter } P \ (\text{take } (k1 + 1) \ xs) = [xs \ ! \ k1]$ 
  proof –
  have  $\text{filter } P \ (\text{take } k1 \ xs) = []$ 
    using  $\langle \forall \ k' < k1. \neg P \ (xs \ ! \ k') \rangle \langle k1 < \text{length } xs \rangle$ 
    using last-filter
    by force
  moreover

```



```

have take (k1 + 1) xs = take k1 xs @ [xs ! k1]
  using ⟨k1 < length xs⟩
  using take-Suc-conv-app-nth
  by auto
ultimately
show ?thesis
  using ⟨P (xs ! k1)⟩
  by simp
qed
then have filter P ?xs ≠ []
  using 0
  by (metis One-nat-def Suc-eq-plus1 append-take-drop-id filter-append length-Cons
length-append less-not-refl3 list.size(3) plus-1-eq-Suc)
  then obtain k2' where *: k2' < length ?xs filter P ?xs ! 0 = ?xs ! k2' P (?xs
! k2') ∨ k' < k2'. ¬ P (?xs ! k')
    using hd-filter[of P ?xs]
    by auto
have filter P xs ! 1 = xs ! (k1 + 1 + k2')
  using * ⟨filter P (take (k1 + 1) xs) = [xs ! k1]⟩ ⟨k1 < length xs⟩
  by (metis One-nat-def Suc-eq-plus1 Suc-leI append-take-drop-id filter-append
length-Cons list.size(3) nth-append-length-plus nth-drop plus-1-eq-Suc)
moreover
have P (xs ! (k1 + 1 + k2'))
  using * ⟨k1 < length xs⟩
  by auto
moreover
have ∀ k'. k1 < k' ∧ k' < k1 + 1 + k2' ⟶ ¬ P (xs ! k')
proof safe
  fix k'
  assume k1 < k' k' < k1 + 1 + k2' P (xs ! k')
  then have k' - (k1 + 1) < k2'
    by auto
  then have ¬ P (?xs ! (k' - (k1 + 1)))
    using ⟨∀ k' < k2'. ¬ P (?xs ! k')⟩
    by simp
  then have ¬ P (xs ! k')
    using ⟨k2' < length ?xs⟩
    using ⟨k1 < k'⟩
    by auto
then show False

```

```

    using ⟨P (xs ! k')⟩
    by simp
qed
moreover
have k1 + 1 + k2' < length xs
  using ⟨k2' < length ?xs⟩
  by auto
ultimately
show ?case
  using ⟨P (xs ! k1)⟩ ⟨filter P xs ! 0 = xs ! k1⟩
  by (rule-tac x=k1 in exI, rule-tac x=k1+1+k2' in exI, simp)
next
case (Suc i)
let ?t = takeWhile (λ x. ¬ P x) xs and ?d = dropWhile (λ x. ¬ P x) xs
let ?xs = tl ?d

have ?xs ≠ []
  using Suc(2)
  by (metis Suc-eq-plus1 add.commute add-less-cancel-left filter.simps(1) filter-dropWhile-not
filter-tl hd-Cons-tl length-Cons list.size(3) not-less-zero)

have *: ∀ k. length ?t + k + 1 < length xs ⟶ xs ! (length ?t + k + 1) = tl
?d ! k
  by (metis One-nat-def add.right-neutral add-Suc-right add-lessD1 hd-Cons-tl
length-append less-le list.size(3) nth-Cons-Suc nth-append-length-plus takeWhile-dropWhile-id)

have i + 1 < length (filter P ?xs)
  using Suc(2)
  by auto
then obtain k1 k2
  where k1 < k2 k2 < length ?xs
    filter P ?xs ! i = ?xs ! k1
    filter P ?xs ! (i + 1) = ?xs ! k2
    P (?xs ! k1)
    P (?xs ! k2)
    ∀ k'. k1 < k' ∧ k' < k2 ⟶ ¬ P (?xs ! k')
  using Suc(1)[of ?xs]
  by auto
show ?case
proof (rule-tac x=k1+length ?t+1 in exI, rule-tac x=k2+length ?t+1 in exI,

```

```

safe)
  show  $k1 + \text{length } ?t + 1 < k2 + \text{length } ?t + 1$ 
    using  $\langle k1 < k2 \rangle$ 
    by simp
next
  have  $k2 + \text{length } ?t + 1 < \text{length } ?xs + 1 + \text{length } ?t$ 
    using  $\langle k2 < \text{length } ?xs \rangle$ 
    by simp
  then show  $k2 + \text{length } ?t + 1 < \text{length } xs$ 
    using  $\langle ?xs \neq [] \rangle$ 
    by (metis One-nat-def Suc-eq-plus1 Suc-pred add.commute add-lessD1 length-append
length-greater-0-conv length-tl less-diff-conv takeWhile-dropWhile-id)
next
  show  $P (xs ! (k1 + \text{length } ?t + 1))$ 
    using  $\langle P (?xs ! k1) \rangle \langle k1 < k2 \rangle \langle k2 < \text{length } ?xs \rangle *$ 
    by (metis Suc-eq-plus1 add.commute add-Suc-right hd-Cons-tl length-greater-0-conv
length-tl list.size(3) not-less-zero nth-Cons-Suc nth-append-length-plus takeWhile-dropWhile-id
zero-less-diff)
next
  show  $P (xs ! (k2 + \text{length } (\text{takeWhile } (\lambda x. \neg P x) xs) + 1))$ 
    using  $\langle P (?xs ! k2) \rangle \langle k2 < \text{length } ?xs \rangle *$ 
    by (metis Suc-eq-plus1 add.commute add-Suc-right hd-Cons-tl length-greater-0-conv
length-tl list.size(3) not-less-zero nth-Cons-Suc nth-append-length-plus takeWhile-dropWhile-id
zero-less-diff)
next
  fix  $k'$ 
  assume  $k1 + \text{length } ?t + 1 < k' k' < k2 + \text{length } ?t + 1 P (xs ! k')$ 
  then have  $k1 < k' - (\text{length } ?t + 1) k' - (\text{length } ?t + 1) < k2$ 
    using  $\langle k1 < k2 \rangle \langle k2 < \text{length } ?xs \rangle$ 
    by linarith+
  moreover
  have  $\text{length } ?t + (k' - (\text{length } ?t + 1)) + 1 < \text{length } xs$ 
    using  $\langle k2 < \text{length } (\text{tl } (\text{dropWhile } (\lambda x. \neg P x) xs)) \rangle$ 
    by (smt ab-semigroup-add-class.add-ac(1) add.commute add-lessD1 add-less-cancel-left
calculation(2) length-append length-tl less-diff-conv less-trans-Suc plus-1-eq-Suc takeWhile-dropWhile-id)
  then have  $P (?xs ! (k' - (\text{length } ?t + 1)))$ 
    using  $*[\text{rule-format, of } k' - (\text{length } ?t + 1)] \langle P (xs ! k') \rangle$ 
    by (metis Suc-eq-plus1 add-Suc add-diff-inverse-nat calculation(1) nat-diff-split
not-less-zero)
  ultimately

```

```

show False
  using  $\langle \forall k'. k1 < k' \wedge k' < k2 \longrightarrow \neg P (?xs ! k') \rangle$  [rule-format, of  $k' - (\text{length } ?t + 1)$ ]  $\langle k1 < k2 \rangle \langle k2 < \text{length } ?xs \rangle$ 
  by simp
next
show  $\text{filter } P \text{ } xs ! (\text{Suc } i) = xs ! (k1 + \text{length } ?t + 1)$ 
proof–
  have  $\text{filter } P \text{ } xs ! (\text{Suc } i) = \text{filter } P \text{ } ?d ! (\text{Suc } i)$ 
  by simp
  also have  $\dots = \text{filter } P \text{ } (\text{tl } ?d) ! i$ 
  using  $\langle ?xs \neq [] \rangle \langle i + 1 < \text{length } (\text{filter } P \text{ } ?xs) \rangle$ 
  by (metis add-lessD1 filter-tl hd-dropWhile list.sel(2) nth-tl)
  finally
show ?thesis
  using  $\langle \text{filter } P \text{ } ?xs ! i = ?xs ! k1 \rangle *$ 
  using  $\langle k1 < k2 \rangle \langle k2 < \text{length } ?xs \rangle$ 
  by (smt Suc-eq-plus1 add.commute add-Suc-right add-lessD1 add-less-cancel-left
length-append length-tl less-diff-conv less-trans-Suc takeWhile-dropWhile-id)
  qed
next
show  $\text{filter } P \text{ } xs ! (\text{Suc } i + 1) = xs ! (k2 + \text{length } ?t + 1)$ 
proof–
  have  $\text{filter } P \text{ } xs ! (\text{Suc } i + 1) = \text{filter } P \text{ } ?d ! (\text{Suc } i + 1)$ 
  by simp
  also have  $\dots = \text{filter } P \text{ } (\text{tl } ?d) ! (\text{Suc } i)$ 
  using  $\langle ?xs \neq [] \rangle \langle i + 1 < \text{length } (\text{filter } P \text{ } ?xs) \rangle$ 
  by (metis add.commute filter-tl hd-dropWhile nth-tl plus-1-eq-Suc tl-Nil)
  finally
show ?thesis
  using  $\langle \text{filter } P \text{ } ?xs ! (i + 1) = ?xs ! k2 \rangle *$ 
  using  $\langle k1 < k2 \rangle \langle k2 < \text{length } ?xs \rangle$ 
  by (smt Suc-eq-plus1 add.commute add-Suc-right add-lessD1 add-less-cancel-left
length-append length-tl less-diff-conv less-trans-Suc takeWhile-dropWhile-id)
  qed
qed
qed

```

Unlabeled states

type-synonym *state* = *nat list*

definition *initial-state* :: *nat* \Rightarrow *state* **where**
initial-state *n* = (*replicate* (*n* + 1) 0) [0 := *n*]

definition *final-state* :: *nat* \Rightarrow *state* **where**
final-state *n* = (*replicate* (*n* + 1) 0) [*n* := *n*]

definition *valid-state* :: *nat* \Rightarrow *state* \Rightarrow *bool* **where**
valid-state *n* *state* \longleftrightarrow *length* *state* = *n* + 1 \wedge *sum-list* *state* = *n*

definition *move* :: *nat* \Rightarrow *nat* \Rightarrow *state* \Rightarrow *state* **where**
move *p1* *p2* *state* =
 (let *k1* = *state* ! *p1*;
 k2 = *state* ! *p2*
 in *state* [*p1* := *k1* - 1, *p2* := *k2* + 1])

definition *valid-move'* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *state* \Rightarrow *state* \Rightarrow *bool* **where**
valid-move' *n* *p1* *p2* *state* *state'* \longleftrightarrow
 (let *k1* = *state* ! *p1*
 in *k1* > 0 \wedge *p1* < *p2* \wedge *p2* \leq *p1* + *k1* \wedge *p2* \leq *n* \wedge
 state' = *move* *p1* *p2* *state*)

definition *valid-move* :: *nat* \Rightarrow *state* \Rightarrow *state* \Rightarrow *bool* **where**
valid-move *n* *state* *state'* \longleftrightarrow
 (\exists *p1* *p2*. *valid-move'* *n* *p1* *p2* *state* *state'*)

definition *valid-moves* **where**
valid-moves *n* *states* \longleftrightarrow
 (\forall *i* < *length* *states* - 1. *valid-move* *n* (*states* ! *i*) (*states* ! (*i* + 1)))

definition *valid-game* **where**
valid-game *n* *states* \longleftrightarrow
 length *states* \geq 2 \wedge
 hd *states* = *initial-state* *n* \wedge
 last *states* = *final-state* *n* \wedge
 valid-moves *n* *states*

lemma *valid-state-initial-state* [*simp*]:
shows *valid-state* *n* (*initial-state* *n*)

by (simp add: initial-state-def valid-state-def)

lemma *valid-move-valid-state*:

assumes *valid-state n state valid-move n state state'*

shows *valid-state n state'*

proof–

obtain *p1 p2*

where $*$: $0 < \text{state} ! p1$ $p1 < p2$ $p2 \leq p1 + \text{state} ! p1$ $p2 \leq n$ $\text{state}' = \text{state}[p1 := \text{state} ! p1 - 1, p2 := \text{state} ! p2 + 1]$

using *assms*

unfolding *valid-move-def valid-move'-def move-def Let-def*

by *auto*

then have *sum-list state > 0*

using *assms(1) valid-state-def*

by *auto*

then have *sum-list (state[p1 := state ! p1 - 1, p2 := state ! p2 + 1]) = sum-list state*

using $*$ *assms*

using *sum-list-update[of p1 state state ! p1 - 1]*

using *sum-list-update[of p2 state[p1 := state ! p1 - 1] state ! p2 + 1]*

unfolding *valid-state-def*

by *auto*

then show *?thesis*

using $\langle \text{valid-state } n \text{ state} \rangle *$

by (simp add: valid-state-def)

qed

lemma *valid-moves-Nil* [simp]:

shows *valid-moves n []*

by (simp add: valid-moves-def)

lemma *valid-moves-Single* [simp]:

shows *valid-moves n [state]*

by (simp add: valid-moves-def)

lemma *valid-moves-Cons* [simp]:

shows *valid-moves n (state1 # state2 # states) \longleftrightarrow*

valid-move n state1 state2 \wedge valid-moves n (state2 # states)

unfolding *valid-moves-def*

by (auto simp add: nth-Cons split: nat.split)

```

lemma valid-moves-valid-states:
  assumes valid-moves n states valid-state n (hd states)
  shows  $\forall$  state  $\in$  set states. valid-state n state
  using assms
proof (induction states)
  case Nil
  then show ?case
    by simp
next
  case (Cons a states)
  then show ?case
    by (metis list.sel(1) list.set-cases set-ConsD valid-moves-Cons valid-move-valid-state)
qed

```

```

lemma valid-game-valid-states:
  assumes valid-game n states
  shows  $\forall$  state  $\in$  set states. valid-state n state
  using assms
  unfolding valid-game-def
  using valid-moves-valid-states
  by fastforce

```

```

definition move-positions where
  move-positions state state' =
    (THE (p1, p2). valid-move' (length state - 1) p1 p2 state state')

```

```

lemma move-positions-unique:
  assumes valid-state n state valid-move n state state'
  shows  $\exists!$  (p1, p2). valid-move' n p1 p2 state state'
proof–
  have length state = n + 1
    using assms
    unfolding valid-state-def
    by simp

  have  $\exists!$  p1. p1 < length state  $\wedge$  state ! p1 > 0  $\wedge$  state' ! p1 = state ! p1 - 1
    using assms
    unfolding valid-state-def valid-move-def valid-move'-def Let-def move-def
    by (smt add.right-neutral add-Suc-right add-diff-cancel-left' le-SucI less-imp-Suc-add)

```

less-le-trans list-update-swap n-not-Suc-n nat.simps(3) nth-list-update-eq nth-list-update-neq plus-1-eq-Suc)

then have *: $\exists! p1. p1 \leq n \wedge state ! p1 > 0 \wedge state' ! p1 = state ! p1 - 1$
using $\langle length\ state = n + 1 \rangle$
by (*metis Nat.le-diff-conv2 Suc-leI add.commute add-diff-cancel-right' le-add2 le-imp-less-Suc plus-1-eq-Suc*)

have $\exists! p2. p2 < length\ state \wedge state' ! p2 = state ! p2 + 1$
using *assms*
unfolding *valid-state-def valid-move-def valid-move'-def Let-def move-def*
by (*metis Groups.add-ac(2) diff-le-self le-imp-less-Suc length-list-update n-not-Suc-n nat-neq-iff nth-list-update-eq nth-list-update-neq plus-1-eq-Suc*)
then have **: $\exists! p2. p2 \leq n \wedge state' ! p2 = state ! p2 + 1$
using $\langle length\ state = n + 1 \rangle$
by (*simp add: discrete*)

obtain *p1 p2* **where** *valid-move' n p1 p2 state state'*

using *assms*

unfolding *valid-move-def*

by *auto*

show *?thesis*

proof

show *case (p1, p2) of (p1, p2) \Rightarrow valid-move' n p1 p2 state state'*

using $\langle valid-move' n p1 p2 state state' \rangle$

by *simp*

next

fix *x*

assume *case x of (p1', p2') \Rightarrow valid-move' n p1' p2' state state'*

then obtain *p1' p2'* **where** *x = (p1', p2') valid-move' n p1' p2' state state'*

by *auto*

then show *x = (p1, p2)*

using $\langle valid-move' n p1 p2 state state' \rangle * ** \langle length\ state = n + 1 \rangle$

unfolding *valid-move'-def move-def Let-def*

by (*metis Nat.add-0-right One-nat-def add-Suc-right le-imp-less-Suc le-less-trans length-list-update less-imp-le-nat nat-neq-iff nth-list-update-eq nth-list-update-neq*)

qed

qed

lemma *valid-move'-move-positions:*

assumes *valid-state n state valid-move' n p1 p2 state state'*


```

shows (p1, p2) = move-positions state state'
proof –
  have *: (THE x. let (p1', p2') = x in valid-move' (length state - 1) p1' p2'
state state') = (p1, p2)
  proof (rule the-equality)
    show let (p1', p2') = (p1, p2) in valid-move' (length state - 1) p1' p2' state
state'
    using assms
    unfolding valid-state-def valid-move-def Let-def
    by auto
  next
    fix x
    assume let (p1', p2') = x in valid-move' (length state - 1) p1' p2' state state'
    then show x = (p1, p2)
      using move-positions-unique[of n state state'] assms
      unfolding valid-state-def valid-move-def
      by auto
    qed
  then show ?thesis
    unfolding move-positions-def Let-def
    by auto
qed

```

```

lemma move-positions-valid-move':
  assumes valid-state n state valid-move n state state'
    (p1, p2) = move-positions state state'
  shows valid-move' n p1 p2 state state'
  using assms
  by (metis fstI sndI valid-move-def valid-move'-move-positions)

```

Labeled states

```

type-synonym labeled-state = (nat set) list

```

```

definition initial-labeled-state :: nat  $\Rightarrow$  labeled-state where
  initial-labeled-state n = (replicate (n+1) {}) [0 := {0.. $n$ }]

```

```

definition final-labeled-state :: nat  $\Rightarrow$  labeled-state where
  final-labeled-state n = (replicate (n+1) {}) [n := {0.. $n$ }]

```

definition *valid-labeled-state* :: *nat* \Rightarrow *labeled-state* \Rightarrow *bool* **where**

valid-labeled-state *n l-state* \longleftrightarrow
 $\text{length } l\text{-state} = n+1 \wedge$
 $(\forall i\ j. i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\}) \wedge$
 $(\bigcup (\text{set } l\text{-state})) = \{0..<n\}$

definition *labeled-move* **where**

labeled-move *p1 p2 stone l-state* =
 $(\text{let } ss1 = l\text{-state} ! p1;$
 $ss2 = l\text{-state} ! p2$
 $\text{in } l\text{-state } [p1 := ss1 - \{stone\}, p2 := ss2 \cup \{stone\}])$

definition *valid-labeled-move'* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *labeled-state* \Rightarrow *labeled-state* \Rightarrow *bool* **where**

valid-labeled-move' *n p1 p2 stone l-state l-state'* \longleftrightarrow
 $(\text{let } ss1 = l\text{-state} ! p1$
 $\text{in } p1 < p2 \wedge p2 \leq p1 + \text{card } ss1 \wedge p2 \leq n \wedge$
 $\text{stone} \in ss1 \wedge l\text{-state}' = \text{labeled-move } p1 p2 \text{ stone } l\text{-state})$

definition *valid-labeled-move* :: *nat* \Rightarrow *labeled-state* \Rightarrow *labeled-state* \Rightarrow *bool* **where**

valid-labeled-move *n l-state l-state'* \longleftrightarrow
 $(\exists p1\ p2\ \text{stone}. \text{valid-labeled-move}'\ n\ p1\ p2\ \text{stone } l\text{-state } l\text{-state}')$

definition *valid-labeled-moves* **where**

valid-labeled-moves *n l-states* \longleftrightarrow
 $(\forall i < \text{length } l\text{-states} - 1. \text{valid-labeled-move } n\ (l\text{-states} ! i)\ (l\text{-states} ! (i + 1)))$

definition *valid-labeled-game* **where**

valid-labeled-game *n l-states* \longleftrightarrow
 $\text{length } l\text{-states} \geq 2 \wedge$
 $\text{hd } l\text{-states} = \text{initial-labeled-state } n \wedge$
 $\text{last } l\text{-states} = \text{final-labeled-state } n \wedge$
 $\text{valid-labeled-moves } n\ l\text{-states}$

lemma *valid-labeled-state-initial-labeled-state* [simp]:

shows *valid-labeled-state* *n* (*initial-labeled-state* *n*)

unfolding *valid-labeled-state-def* *initial-labeled-state-def*

by *auto*

lemma *valid-labeled-state-final-labeled-state* [simp]:
shows *valid-labeled-state* n (*final-labeled-state* n)
proof–
have (*replicate* (*Suc* n) $\{\}$)[$n := \{0..<n\}$] = (*replicate* n $\{\}$) @ [$\{0..<n\}$]
by (*metis* *length-replicate* *list-update-length* *replicate-Suc* *replicate-append-same*)
then show ?thesis
unfolding *valid-labeled-state-def* *final-labeled-state-def*
by (*auto* *simp* *del: replicate-Suc* *simp* *add: nth-append*)
qed

lemma *valid-labeled-move-valid-labeled-state*:
assumes *valid-labeled-state* n *l-state* *valid-labeled-move* n *l-state* *l-state'*
shows *valid-labeled-state* n *l-state'*
proof–
from *assms* **obtain** $p1$ $p2$ *stone* **where**
 $**:$ $p1 < p2$ $p2 \leq p1 + \text{card } (l\text{-state} ! p1)$ $p2 \leq n$ $\text{length } l\text{-state} = n+1 \cup$
 $(\text{set } l\text{-state}) = \{0..<n\} \forall i j. i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\}$
 $\text{stone} \in l\text{-state} ! p1$ $l\text{-state}' = l\text{-state}[p1 := l\text{-state} ! p1 - \{\text{stone}\}, p2 := l\text{-state}$
 $! p2 \cup \{\text{stone}\}]$
unfolding *valid-labeled-move-def* *valid-labeled-move'-def* *valid-labeled-state-def*
Let-def *labeled-move-def*
by *auto*

then have $*$: $\forall i \leq n. l\text{-state}' ! i = (\text{if } i = p1 \text{ then } l\text{-state} ! p1 - \{\text{stone}\}$
 $\text{else if } i = p2 \text{ then } l\text{-state} ! p2 \cup \{\text{stone}\}$
 $\text{else } l\text{-state} ! i)$ $\text{length } l\text{-state}' = n + 1$
by *auto*

have $\text{stone} \notin l\text{-state} ! p2$
using $\langle \forall i j. i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\} \rangle \langle \text{stone} \in l\text{-state}$
 $! p1 \rangle$
using $\langle p1 < p2 \rangle \langle p2 \leq n \rangle$
by (*metis* *Collect-mem-eq* *IntI* *empty-Collect-eq*)

have $\forall i \leq n. i \neq p1 \longrightarrow \text{stone} \notin l\text{-state} ! i$
using $\langle \forall i j. i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\} \rangle \langle \text{stone} \in l\text{-state}$
 $! p1 \rangle$
using $\langle p1 < p2 \rangle \langle p2 \leq n \rangle$
by (*metis* *disjoint-iff-not-equal* *le-less-trans* *less-imp-le-nat* *nat-neq-iff*)

```

have  $\bigcup (set\ l-state') = \bigcup (set\ l-state)$ 
proof safe
  fix  $x\ X$ 
  assume  $x \in X\ X \in set\ l-state'$ 
  then obtain  $i$  where  $x \in l-state' ! i\ i \leq n$ 
    using  $\langle length\ l-state' = n+1 \rangle$ 
    by (metis One-nat-def add.right-neutral add-Suc-right in-set-conv-nth le-simps(2))

  then show  $x \in \bigcup (set\ l-state)$ 
    using  $*\langle stone \in l-state ! p1 \rangle **$ 
    by (smt Diff-iff One-nat-def Un-insert-right add.right-neutral add-Suc-right
boolean-algebra-cancel.sup0 insertE le-imp-less-Suc le-less-trans less-imp-le-nat mem-simps(9)
nth-mem)
  next
    fix  $x\ X$ 
    assume  $x \in X\ X \in set\ l-state$ 
    then obtain  $i$  where  $i \leq n\ x \in l-state ! i$ 
      using  $\langle length\ l-state = n + 1 \rangle$ 
      by (metis add.commute in-set-conv-nth le-simps(2) plus-1-eq-Suc)
    show  $x \in \bigcup (set\ l-state')$ 
    proof (cases i = p1)
      case True
        then have  $x \in l-state' ! p1 \vee x \in l-state' ! p2$ 
          using  $*\langle p1 < p2 \rangle \langle p2 \leq n \rangle \langle x \in l-state ! i \rangle$ 
          by auto
        then show ?thesis
          using  $\langle p1 < p2 \rangle \langle p2 \leq n \rangle$ 
          using  $*(2)\ mem-simps(9)\ nth-mem$ 
          by auto
      case False
        then have  $x \in l-state' ! i$ 
          using  $*\langle p1 < p2 \rangle \langle p2 \leq n \rangle \langle x \in l-state ! i \rangle$ 
          using  $\langle i \leq n \rangle$  by auto
        then show ?thesis
          by (metis *(2) One-nat-def Sup-upper  $\langle i \leq n \rangle$  add.right-neutral add-Suc-right
le-imp-less-Suc nth-mem subsetD)
    qed
  qed

```

moreover

```

have  $\forall i j. i < j \wedge j \leq n \longrightarrow l\text{-state}' ! i \cap l\text{-state}' ! j = \{\}$ 
proof safe
  fix  $i j x$ 
  assume ***:  $i < j \wedge j \leq n \wedge x \in l\text{-state}' ! i \wedge x \in l\text{-state}' ! j$ 
  then have False
    using *  $\langle \forall i j. i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\} \rangle$ 
    using  $\langle \text{stone} \in l\text{-state} ! p1 \rangle \langle \text{stone} \notin l\text{-state} ! p2 \rangle \langle \forall i \leq n. i \neq p1 \longrightarrow \text{stone} \notin l\text{-state} ! i \rangle$ 
    using  $\langle \text{length } l\text{-state} = n+1 \rangle \langle \text{length } l\text{-state}' = n+1 \rangle \langle p1 < p2 \rangle \langle p2 \leq n \rangle$ 
    apply (cases  $j = p2$ )
    apply (smt Diff-insert-absorb Diff-subset IntI Un-insert-right boolean-algebra-cancel.sup0
empty-iff insertE less-imp-le-nat less-le-trans mk-disjoint-insert nat-neq-iff subsetD)
    apply (smt Un-insert-right boolean-algebra-cancel.sup0 disjoint-iff-not-equal
insert-Diff insert-iff less-imp-le-nat less-le-trans)
  done
  then show  $x \in \{\}$ 
  by simp
qed

```

ultimately

```

show ?thesis
  unfolding valid-labeled-state-def
  using assms
  unfolding valid-labeled-move-def Let-def valid-labeled-move'-def labeled-move-def
valid-labeled-state-def
  by auto
qed

```

lemma valid-labeled-moves-valid-labeled-states:

```

assumes valid-labeled-moves  $n$  l-states valid-labeled-state  $n$  (hd l-states)
shows  $\forall \text{state} \in \text{set } l\text{-states}. \text{valid-labeled-state } n \text{ state}$ 
using assms
proof (induction l-states)
  case Nil
  then show ?case
  by simp

```

```

next
  case (Cons a states)
  then show ?case
    by (metis (no-types, lifting) Groups.add-ac(2) hd-Cons-tl length-greater-0-conv
length-tl less-diff-conv list.inject list.set-cases list.simps(3) nth-Cons-0 nth-Cons-Suc
plus-1-eq-Suc valid-labeled-moves-def valid-labeled-move-valid-labeled-state)
qed

```

```

lemma valid-labeled-game-valid-labeled-states:
  assumes valid-labeled-game n states
  shows  $\forall$  state  $\in$  set states. valid-labeled-state n state
  using assms
  unfolding valid-labeled-game-def
  using valid-labeled-moves-valid-labeled-states
  by fastforce

```

```

definition labeled-move-positions where
  labeled-move-positions state state' =
    (THE (p1, p2, stone). valid-labeled-move' (length state - 1) p1 p2 stone
state state')

```

```

lemma labeled-move-positions-unique:
  assumes valid-labeled-state n state valid-labeled-move n state state'
  shows  $\exists!$  (p1, p2, stone). valid-labeled-move' n p1 p2 stone state state'
proof-
  obtain p1 p2 stone where *: valid-labeled-move' n p1 p2 stone state state'
  using assms
  unfolding valid-labeled-move-def
  by auto
  show ?thesis
proof
  show case (p1, p2, stone) of (p1, p2, stone)  $\Rightarrow$  valid-labeled-move' n p1 p2
stone state state'
    using *
    by auto
next
  fix x :: nat  $\times$  nat  $\times$  nat
  obtain p1' p2' stone' where x: x = (p1', p2', stone')
  by (cases x)
  assume case x of (p1, p2, stone)  $\Rightarrow$  valid-labeled-move' n p1 p2 stone state

```

```

state'
  then have **: valid-labeled-move' n p1' p2' stone' state state'
    using x
    by simp
  have *: p1 < p2 p2 ≤ n stone < n stone ∈ state ! p1 stone ∉ state' ! p1 stone
    ∉ state ! p2 stone ∈ state' ! p2
    ∀ stone'' p. p ≤ n ∧ stone'' < n ∧ stone'' ≠ stone → (stone'' ∈ state !
p ↔ stone'' ∈ state' ! p)
    using * assms(1)
  unfolding valid-labeled-state-def valid-labeled-move'-def Let-def labeled-move-def
  by (auto simp add: nth-list-update)

  have **: p1' < p2' p2' ≤ n stone' < n stone' ∈ state ! p1' stone' ∉ state' !
p1' stone' ∉ state ! p2' stone' ∈ state' ! p2'
    ∀ stone'' p. p ≤ n ∧ stone'' < n ∧ stone'' ≠ stone' → (stone'' ∈ state !
p ↔ stone'' ∈ state' ! p)
    using ** assms(1)
  unfolding valid-labeled-state-def valid-labeled-move'-def Let-def labeled-move-def
  by (auto simp add: nth-list-update)

  have stone = stone'
    using * **
    by auto

  have disj: ∀ i j. i < j ∧ j ≤ n → state ! i ∩ state ! j = {}
    using assms(1)
    unfolding valid-labeled-state-def
    by auto

  have p1 = p1'
    using *(4) *(4) ⟨stone = stone'⟩ *(1-2) *(1-2)
    using disj[rule-format, of p1 p1']
    using disj[rule-format, of p1' p1]
    by force

  have valid-labeled-state n state'
    using assms(1) assms(2) valid-labeled-move-valid-labeled-state by blast
  then have disj': ∀ i j. i < j ∧ j ≤ n → state' ! i ∩ state' ! j = {}
    unfolding valid-labeled-state-def
    by auto

```

```

have p2 = p2'
  using *(7) ***(7) ⟨stone = stone'⟩ *(2) ***(2)
  using disj'[rule-format, of p2 p2']
  using disj'[rule-format, of p2' p2]
  by force

then show x = (p1, p2, stone)
  using x ⟨stone = stone'⟩ ⟨p1 = p1'⟩ ⟨p2 = p2'⟩
  by auto
qed
qed

```

lemma *labeled-move-positions*:

```

assumes valid-labeled-state n state valid-labeled-move' n p1 p2 stone state state'
shows labeled-move-positions state state' = (p1, p2, stone)
using assms
using labeled-move-positions-unique[OF assms(1), of state']
unfolding labeled-move-positions-def valid-labeled-state-def valid-labeled-move-def
by auto (smt case-prodI the-equality)

```

lemma *labeled-move-positions-valid-move'*:

```

assumes valid-labeled-state n state valid-labeled-move n state state'
          labeled-move-positions state state' = (p1, p2, stone)
shows valid-labeled-move' n p1 p2 stone state state'
using assms(1) assms(2) assms(3) labeled-move-positions valid-labeled-move-def
by auto

```

definition *stone-position* :: *labeled-state* \Rightarrow *nat* \Rightarrow *nat* **where**

```

stone-position l-state stone =
  (THE k. k < length l-state  $\wedge$  stone  $\in$  l-state ! k)

```

lemma *stone-position-unique*:

```

assumes valid-labeled-state n l-state stone < n
shows  $\exists!$  k. k < length l-state  $\wedge$  stone  $\in$  l-state ! k

```

proof—

```

from assms have stone  $\in \bigcup$  (set l-state)
  unfolding valid-labeled-state-def
  by auto
then obtain k where *: k < length l-state stone  $\in$  l-state ! k

```



```

    by (metis UnionE in-set-conv-nth)
  then have  $\forall k'. k' < \text{length } l\text{-state} \wedge \text{stone} \in l\text{-state} \rightarrow k' = k$ 
    using assms
    unfolding valid-labeled-state-def
    by (metis IntI Suc-eq-plus1 empty-iff le-simps(2) nat-neq-iff)
  then show ?thesis
    using *
    by auto
qed

```

lemma *stone-position*:

```

  assumes valid-labeled-state n l-state stone < n
  shows stone-position l-state stone  $\leq n \wedge$ 
        stone  $\in l\text{-state} \rightarrow (\text{stone-position } l\text{-state } \text{stone})$ 
  using assms stone-position-unique[OF assms]
  using theI[of  $\lambda k. k < \text{length } l\text{-state} \wedge \text{stone} \in l\text{-state} \rightarrow k$ ]
  unfolding valid-labeled-state-def stone-position-def
  by (metis (mono-tags, lifting) One-nat-def add.right-neutral add-Suc-right le-simps(2))

```

lemma *stone-positionI*:

```

  assumes valid-labeled-state n l-state stone < n
        k < length l-state stone  $\in l\text{-state} \rightarrow k$ 
  shows stone-position l-state stone = k
  unfolding stone-position-def
  using assms stone-position-unique
  by blast

```

lemma *valid-labeled-move'-stone-positions*:

```

  assumes valid-labeled-state n l-state valid-labeled-move' n p1 p2 stone l-state
        l-state'
  shows stone-position l-state stone = p1  $\wedge$  stone-position l-state' stone = p2
  proof safe
    show stone-position l-state stone = p1
    proof (rule stone-positionI)
      show valid-labeled-state n l-state stone < n p1 < length l-state stone  $\in l\text{-state}$ 
        ! p1
      using assms
      unfolding valid-labeled-state-def valid-labeled-move'-def Let-def
      by auto
    qed
  qed

```

next

show *stone-position l-state' stone = p2*

proof (*rule stone-positionI*)

show *valid-labeled-state n l-state'*

using *assms(1) assms(2) valid-labeled-move-def valid-labeled-move-valid-labeled-state*
by *blast*

next

show *stone < n p2 < length l-state' stone ∈ l-state' ! p2*

using *assms*

unfolding *valid-labeled-state-def valid-labeled-move'-def Let-def labeled-move-def*
by *auto*

qed

qed

lemma *valid-labeled-move'-stone-positions-other:*

assumes *valid-labeled-state n l-state valid-labeled-move' n p1 p2 stone l-state l-state'*

shows $\forall \text{ stone}'. \text{stone}' \neq \text{stone} \wedge \text{stone}' < n \longrightarrow$

$\text{stone-position } l\text{-state}' \text{ stone}' = \text{stone-position } l\text{-state } \text{stone}'$

proof *safe*

fix *stone'*

assume *stone' < n stone' ≠ stone*

show *stone-position l-state' stone' = stone-position l-state stone'*

proof (*rule stone-positionI*)

show *stone' < n*

by *fact*

next

show *valid-labeled-state n l-state'*

using *assms*

using *valid-labeled-move-def valid-labeled-move-valid-labeled-state*

by *blast*

next

show *stone-position l-state stone' < length l-state'*

using $\langle \text{stone}' < n \rangle \text{ assms}(1-2) \text{ stone-position}[\text{of } n \text{ l-state } \text{stone}']$

unfolding *valid-labeled-state-def*

by (*metis Suc-eq-plus1 labeled-move-def le-imp-less-Suc length-list-update valid-labeled-move'-def*)

next

show *stone' ∈ l-state' ! stone-position l-state stone'*

proof—

```

have  $stone' \in l\text{-state} \mid stone\text{-position } l\text{-state } stone'$ 
   $stone\text{-position } l\text{-state } stone' < length\ l\text{-state}$ 
using  $\langle stone' < n \rangle\ assms(1-2)\ stone\text{-position}[of\ n\ l\text{-state } stone']$ 
unfolding  $valid\text{-labeled-state-def}$ 
by auto
then show ?thesis
  using  $\langle stone' \neq stone \rangle\ \langle valid\text{-labeled-move}'\ n\ p1\ p2\ stone\ l\text{-state } l\text{-state}' \rangle$ 
  unfolding  $valid\text{-labeled-move}'\text{-def}\ labeled\text{-move-def}\ Let\text{-def}$ 
  by (metis (no-types, lifting) Un-insert-right boolean-algebra-cancel.sup0
insert-Diff insert-iff length-list-update nth-list-update-eq nth-list-update-neq)
  qed
qed
qed

```

Unlabel

definition $unlabel :: labeled\text{-state} \Rightarrow state$ **where**
 $unlabel = map\ card$

lemma $unlabel\text{-initial}$ [*simp*]:
shows $unlabel\ (initial\text{-labeled-state } n) = initial\text{-state } n$
unfolding $initial\text{-labeled-state-def}\ initial\text{-state-def}\ unlabel\text{-def}$
by *auto*

lemma $unlabel\text{-final}$ [*simp*]:
shows $unlabel\ (final\text{-labeled-state } n) = final\text{-state } n$
unfolding $final\text{-labeled-state-def}\ final\text{-state-def}\ unlabel\text{-def}$
by (*metis* *card-atLeastLessThan* *card-empty* *diff-zero* *map-replicate* *map-update*)

lemma $unlabel\text{-valid}$:
assumes $valid\text{-labeled-state } n\ l\text{-state}$
shows $valid\text{-state } n\ (unlabel\ l\text{-state})$
unfolding $valid\text{-state-def}\ unlabel\text{-def}$

proof

```

let  $?state = map\ card\ l\text{-state}$ 
show  $length\ ?state = n + 1$ 
  using assms
  by (simp add:  $valid\text{-labeled-state-def}$ )

```

show $sum\text{-list } ?state = n$

proof–

let $?s = \text{filter } (\lambda y. \text{card } y \neq 0) \text{ } l\text{-state}$

have $(\sum x \leftarrow l\text{-state}. \text{card } x) = (\sum x \leftarrow ?s. \text{card } x)$

by $(\text{metis } (\text{mono-tags}, \text{lifting}) \text{ sum-list-map-filter})$

also have $\dots = (\sum x \in \text{set } ?s. \text{card } x)$

proof–

have $\forall i j. i < j \wedge j < \text{length } l\text{-state} \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\}$

using assms

unfolding $\text{valid-labeled-state-def}$

by simp

then have $\text{distinct } ?s$

proof $(\text{induction } l\text{-state})$

case Nil

then show $?case$

by simp

next

case $(\text{Cons } a \text{ } l\text{-state})$

have $\forall i j. i < j \wedge j < \text{length } l\text{-state} \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\}$

using $\text{Cons}(2)$

by $(\text{metis } \text{One-nat-def } \text{Suc-eq-plus1 } \text{Suc-less-eq } \text{list.size}(4) \text{ } \text{nth-Cons-Suc})$

then have $\text{distinct } (\text{filter } (\lambda y. \text{card } y \neq 0) \text{ } l\text{-state})$

using $\text{Cons}(1)$

by simp

moreover

have $\text{card } a > 0 \longrightarrow a \notin \text{set } l\text{-state}$

proof safe

assume $\text{card } a > 0 \text{ } a \in \text{set } l\text{-state}$

show False

using $\text{Cons}(2)[\text{rule-format}, \text{of } 0] \langle 0 < \text{card } a \rangle \langle a \in \text{set } l\text{-state} \rangle$

by $(\text{metis } \text{card-empty } \text{in-set-conv-nth } \text{inf.idem } \text{le-simps}(2) \text{ } \text{length-Cons}$

$\text{not-le } \text{nth-Cons-0 } \text{nth-Cons-Suc } \text{zero-less-Suc})$

qed

ultimately

show $?case$

using Cons

by auto

qed

then show $?thesis$

using $\text{sum-list-distinct-conv-sum-set}$ **by** blast

```

qed
also have ... = card (⋃ (set ?s))
proof-
  have  $\forall i \in \text{set } ?s. \text{finite } (id\ i)$ 
    using assms
    unfolding valid-labeled-state-def
    by fastforce
  moreover
  have  $\forall i \in \text{set } ?s. \forall j \in \text{set } ?s. i \neq j \longrightarrow id\ i \cap id\ j = \{\}$ 
  proof (rule ballI, rule ballI, rule impI)
    fix  $i\ j$ 
    assume  $i \in \text{set } ?s\ j \in \text{set } ?s\ i \neq j$ 
    then obtain  $i'\ j'$  where  $i = l\text{-state } !\ i'\ j = l\text{-state } !\ j'\ i' \leq n\ j' \leq n$ 
      using assms
      unfolding valid-labeled-state-def
    by (metis Suc-eq-plus1 filter-is-subset in-set-conv-nth le-simps(2) subsetD)
    then show  $id\ i \cap id\ j = \{\}$ 
      using assms  $\langle i \neq j \rangle$ 
      unfolding valid-labeled-state-def
      by (metis disjoint-iff-not-equal id-apply nat-neq-iff)
  qed
ultimately
show ?thesis
  using card-UN-disjoint[of set ?s id, symmetric]
  by simp
qed
also have  $card\ (\bigcup (set\ ?s)) = card\ (\bigcup (set\ l\text{-state}))$ 
proof-
  have  $\bigcup (set\ ?s) = \bigcup (set\ l\text{-state})$ 
  proof
    show  $\bigcup (set\ l\text{-state}) \subseteq \bigcup (set\ ?s)$ 
    proof safe
      fix  $x\ X$ 
      assume *:  $x \in X\ X \in set\ l\text{-state}$ 
      then have  $card\ X \neq 0$ 
        using assms
        unfolding valid-labeled-state-def
        using Union-upper finite-subset
        by fastforce

```

```

    then show  $x \in \bigcup (set \ ?s)$ 
      using *
      by auto
    qed
  qed auto
  then show ?thesis
    by simp
  qed
finally
show ?thesis
  using assms
  unfolding valid-labeled-state-def
  by simp
qed
qed

```

lemma *unlabel-valid-move'*:

assumes *valid-labeled-state* *n* *l-state* *valid-labeled-move'* *n* *p1* *p2* *stone* *l-state* *l-state'*

shows *valid-move'* *n* *p1* *p2* (*unlabel* *l-state*) (*unlabel* *l-state'*) \wedge
unlabel *l-state'* = *move* *p1* *p2* (*unlabel* *l-state*)

proof—

from *assms* **have**

$p1 < p2$ $p2 \leq p1 + \text{card } (l\text{-state} ! p1)$ $p2 \leq n$ $\text{length } l\text{-state} = n+1$ $\bigcup (set \ l\text{-state}) = \{0..<n\} \ \forall i \ j. \ i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\}$
 $stone \in l\text{-state} ! p1$ $l\text{-state}' = l\text{-state}[p1 := l\text{-state} ! p1 - \{stone\}, p2 := l\text{-state} ! p2 \cup \{stone\}]$

unfolding *valid-labeled-move-def* *valid-labeled-move'-def* *valid-labeled-state-def* *unlabel-def* *Let-def* *labeled-move-def*
by *auto*

have *finite* (*l-state* ! *p1*) \wedge *finite* (*l-state* ! *p2*)

using $\langle \bigcup (set \ l\text{-state}) = \{0..<n\} \rangle$

using $\langle \text{length } l\text{-state} = n + 1 \rangle$ $\langle p1 < p2 \rangle$ $\langle p2 \leq n \rangle$

by (*metis* *One-nat-def* *Union-upper* *add.right-neutral* *add-Suc-right* *finite-atLeastLessThan* *finite-subset* *le-imp-less-Suc* *le-less-trans* *less-imp-le-nat* *nth-mem*)

have $stone \notin l\text{-state} ! p2$

using $\langle stone \in l\text{-state} ! p1 \rangle \langle \forall i \ j. \ i < j \wedge j \leq n \longrightarrow l\text{-state} ! i \cap l\text{-state} ! j = \{\} \rangle$

```

using  $\langle \text{length } l\text{-state} = n + 1 \rangle \langle p1 < p2 \rangle \langle p2 \leq n \rangle$ 
by (metis Collect-empty-eq Collect-mem-eq IntI)

have  $\text{card } (l\text{-state} ! p1) > 0 \text{ length } l\text{-state}' = \text{length } l\text{-state}$ 
 $\text{card } (l\text{-state}' ! p1) = \text{card } (l\text{-state} ! p1) - 1 \text{ card } (l\text{-state}' ! p2) = \text{card}$ 
 $(l\text{-state} ! p2) + 1$ 
 $\forall p. p \leq n \wedge p \neq p1 \wedge p \neq p2 \longrightarrow \text{card } (l\text{-state}' ! p) = \text{card } (l\text{-state} ! p)$ 
using  $\langle \text{finite } (l\text{-state} ! p1) \wedge \text{finite } (l\text{-state} ! p2) \rangle \langle \text{stone} \in l\text{-state} ! p1 \rangle$ 
using  $\langle \text{stone} \notin l\text{-state} ! p2 \rangle \langle l\text{-state}' = l\text{-state}[p1 := l\text{-state} ! p1 - \{\text{stone}\}, p2$ 
 $:= l\text{-state} ! p2 \cup \{\text{stone}\}] \rangle$ 
using  $\langle \text{length } l\text{-state} = n + 1 \rangle \langle p1 < p2 \rangle \langle p2 \leq n \rangle$ 
using card-0-eq
by - (blast, simp+)

then show ?thesis
using  $\langle \text{length } l\text{-state} = n + 1 \rangle \langle p1 < p2 \rangle \langle p2 \leq p1 + \text{card } (l\text{-state} ! p1) \rangle \langle p2$ 
 $\leq n \rangle$ 
using  $\langle l\text{-state}' = l\text{-state}[p1 := l\text{-state} ! p1 - \{\text{stone}\}, p2 := l\text{-state} ! p2 \cup$ 
 $\{\text{stone}\}] \rangle$ 
unfolding unlabel-def valid-move'-def
by (auto simp add: move-def map-update)
qed

```

lemma *unlabel-valid-move:*

```

assumes valid-labeled-state  $n \text{ } l\text{-state} \text{ } \text{valid-labeled-move } n \text{ } l\text{-state} \text{ } l\text{-state}'$ 
shows valid-move  $n \text{ } (\text{unlabel } l\text{-state}) \text{ } (\text{unlabel } l\text{-state}')$ 
using assms(2) unlabel-valid-move'[OF assms(1)]
unfolding valid-labeled-move-def valid-move-def Let-def
by force

```

Labeled move max stone

definition *valid-labeled-move-max-stone* :: $\text{nat} \Rightarrow \text{labeled-state} \Rightarrow \text{labeled-state} \Rightarrow \text{bool}$ **where**

```

valid-labeled-move-max-stone  $n \text{ } l\text{-state} \text{ } l\text{-state}' \longleftrightarrow$ 
 $(\exists p1 \text{ } p2. \text{valid-labeled-move}' \text{ } n \text{ } p1 \text{ } p2 \text{ } (\text{Max } (l\text{-state} ! p1)) \text{ } l\text{-state} \text{ } l\text{-state}')$ 

```

definition *valid-labeled-moves-max-stone* **where**

```

valid-labeled-moves-max-stone  $n \text{ } l\text{-states} \longleftrightarrow$ 
 $(\forall i < \text{length } l\text{-states} - 1. \text{valid-labeled-move-max-stone } n \text{ } (l\text{-states} ! i) \text{ } (l\text{-states}$ 

```

$! (i + 1)))$

definition *valid-labeled-game-max-stone* **where**

valid-labeled-game-max-stone n l -states \longleftrightarrow
 $\text{length } l\text{-states} \geq 2 \wedge$
 $\text{hd } l\text{-states} = \text{initial-labeled-state } n \wedge$
 $\text{last } l\text{-states} = \text{final-labeled-state } n \wedge$
valid-labeled-moves-max-stone n l -states

lemma *valid-labeled-moves-max-stone-Cons*:

assumes *valid-labeled-moves-max-stone* n *states* *valid-labeled-move-max-stone* n
state ($\text{hd } \text{states}$)
shows *valid-labeled-moves-max-stone* n ($\text{state} \# \text{states}$)
using *assms*
using *less-Suc-eq-0-disj*
apply (*cases states*)
apply (*simp add: valid-labeled-moves-max-stone-def*)
apply (*auto simp add: valid-labeled-moves-max-stone-def*)
done

lemma *valid-labeled-game-max-stone-valid-labeled-game*:

assumes *valid-labeled-game-max-stone* n *states*
shows *valid-labeled-game* n *states*
using *assms*
unfolding *valid-labeled-game-max-stone-def*
unfolding *valid-labeled-game-def* *valid-labeled-moves-def* *valid-labeled-moves-max-stone-def*
unfolding *valid-labeled-move-max-stone-def* *valid-labeled-move-def*
by *force*

lemma *valid-labeled-move-move-max-stone*:

assumes *valid-labeled-state* n l -state
 $\text{unlabel } l\text{-state} = \text{state } \text{valid-move}' n p1 p2 \text{ state } \text{state}'$
 $l\text{-state}' = \text{labeled-move } p1 p2 (\text{Max } (l\text{-state} ! p1)) l\text{-state}$
shows *valid-labeled-move'* n $p1 p2 (\text{Max } (l\text{-state} ! p1)) l\text{-state } l\text{-state}'$

proof—

have $\text{Max } (l\text{-state} ! p1) \in l\text{-state} ! p1$

by (*metis (no-types, lifting) Max-in assms(1) assms(2) assms(3) card-empty*
card-infinite less-le-trans nat-neq-iff nth-map trans-less-add1 unlabel-def valid-labeled-state-def
valid-move'-def)

then show *?thesis*


```

using assms
by (metis (no-types, lifting) less-le-trans nth-map trans-less-add1 unlabel-def
valid-labeled-move'-def valid-labeled-state-def valid-move'-def)
qed

```

```

primrec label-moves-max-stone where
  label-moves-max-stone l-state [] = [l-state]
| label-moves-max-stone l-state (state' # states) =
  (let state = unlabel l-state;
    (p1, p2) = move-positions state state';
    l-state' = labeled-move p1 p2 (Max (l-state ! p1)) l-state
    in l-state # label-moves-max-stone l-state' states)

```

```

lemma hd-label-moves-max-stone [simp]:
shows hd (label-moves-max-stone l-state states) = l-state
by (induction states) (auto simp add: Let-def split: prod.split)

```

```

lemma valid-states-label-moves-max-stone:
assumes valid-labeled-state n l-state valid-moves n (unlabel l-state # states)
shows  $\forall$  l-state'  $\in$  set (label-moves-max-stone l-state states). valid-labeled-state
n l-state'

```

```

using assms
proof (induction states arbitrary: l-state)
case Nil
then show ?case
by simp
next
case (Cons state' states)
let ?state = unlabel l-state
let ?p = move-positions ?state state'
let ?p1 = fst ?p
let ?p2 = snd ?p
let ?stone = Max (l-state ! ?p1)
let ?l-state' = labeled-move ?p1 ?p2 ?stone l-state

have valid-state n ?state
using (valid-labeled-state n l-state)
by (simp add: unlabel-valid)

have valid-move n ?state state'

```

```

using Cons(3)
by (metis Groups.add-ac(2) One-nat-def add-diff-cancel-left' add-is-0 grOI
list.size(4) n-not-Suc-n nth-Cons-0 nth-Cons-Suc plus-1-eq-Suc valid-moves-def)

have valid-move' n ?p1 ?p2 ?state state'
using ⟨valid-state n ?state⟩ ⟨valid-move n ?state state'⟩
by (simp add: move-positions-valid-move')

have **: valid-labeled-move' n ?p1 ?p2 ?stone l-state ?l-state'
proof (rule valid-labeled-move-move-max-stone)
  show valid-labeled-state n l-state
  by fact
next
  show unlabel l-state = unlabel l-state
  by simp
next
  show valid-move' n ?p1 ?p2 ?state state'
  by fact
qed simp

have move ?p1 ?p2 ?state = state'
using ⟨valid-move' n ?p1 ?p2 ?state state'⟩
unfolding valid-move'-def Let-def
by simp
then have *: unlabel ?l-state' = state'
using unlabel-valid-move'[OF Cons(2) **, THEN conjunct2]
by simp

have  $\forall$  l-state'  $\in$  set (label-moves-max-stone ?l-state' states). valid-labeled-state
n l-state'
proof (rule Cons(1))
  have valid-labeled-move n l-state ?l-state'
  using **
  unfolding valid-labeled-move-def
  by metis
  then show valid-labeled-state n ?l-state'
  using Cons(2)
  using valid-labeled-move-valid-labeled-state
  by blast
next

```

```

show valid-moves n (unlabel ?l-state' # states)
  using Cons(3) ⟨valid-move n (unlabel l-state) state⟩
  using *
    by (metis (no-types, lifting) One-nat-def add-Suc-right diff-add-inverse2
group-cancel.add1 less-diff-conv list.size(4) nth-Cons-Suc plus-1-eq-Suc valid-moves-def)
  qed
then show ?case
  using Cons(2)
    by (metis (mono-tags, lifting) label-moves-max-stone.simps(2) prod.collapse
prod.simps(2) set-ConsD)
qed

```

lemma *unlabel-label-moves-max-stone:*

```

assumes valid-labeled-state n l-state valid-moves n (unlabel l-state # states)
shows map unlabel (label-moves-max-stone l-state states) = unlabel l-state #
states

```

```

using assms
proof (induction states arbitrary: l-state)
  case Nil
  then show ?case
    by simp
next
  case (Cons state' states)
  let ?state = unlabel l-state
  let ?p = move-positions ?state state'
  let ?p1 = fst ?p
  let ?p2 = snd ?p
  let ?stone = Max (l-state ! ?p1)
  let ?l-state' = labeled-move ?p1 ?p2 ?stone l-state

```

```

have valid-state n ?state
  using ⟨valid-labeled-state n l-state⟩
  by (simp add: unlabel-valid)

```

```

have valid-move n ?state state'
  using Cons(3)
    by (metis Groups.add-ac(2) One-nat-def add-diff-cancel-left' add-is-0 grOI
list.size(4) n-not-Suc-n nth-Cons-0 nth-Cons-Suc plus-1-eq-Suc valid-moves-def)

```

```

have valid-move' n ?p1 ?p2 ?state state'

```

```

using ⟨valid-state n ?state⟩ ⟨valid-move n ?state state'⟩
by (simp add: move-positions-valid-move')

have **: valid-labeled-move' n ?p1 ?p2 ?stone l-state ?l-state'
proof (rule valid-labeled-move-move-max-stone)
  show valid-labeled-state n l-state
  by fact
next
  show unlabel l-state = unlabel l-state
  by simp
next
  show valid-move' n ?p1 ?p2 ?state state'
  by fact
qed simp

have move ?p1 ?p2 ?state = state'
  using ⟨valid-move' n ?p1 ?p2 ?state state'⟩
  unfolding valid-move'-def Let-def
  by simp
then have *: unlabel ?l-state' = state'
  using unlabel-valid-move'[OF Cons(2) **, THEN conjunct2]
  by simp

have map unlabel (label-moves-max-stone ?l-state' states) = unlabel ?l-state' #
states
proof (rule Cons(1))
  show valid-moves n ((unlabel ?l-state') # states)
  using Cons(3) *
  using less-diff-conv valid-moves-def
  by auto
next
  have valid-labeled-move n l-state ?l-state'
  using **
  unfolding valid-labeled-move-def
  by metis
  then show valid-labeled-state n ?l-state'
  using Cons(2)
  using valid-labeled-move-valid-labeled-state
  by blast
qed

```

```

then show ?case
  using * ⟨valid-move' n ?p1 ?p2 (unlabel l-state) state'⟩ ⟨valid-state n (unlabel
l-state)⟩
  by (smt Cons-eq-map-conv case-prod-conv label-moves-max-stone.simps(2) valid-move'-move-positions)
qed

```

```

lemma label-moves-max-stone-length [simp]:
  shows length (label-moves-max-stone l-state states) = length states + 1
  by (induction states arbitrary: l-state) (auto split: prod.split)

```

```

lemma label-moves-max-stone-valid-moves:
  assumes valid-labeled-state n l-state valid-moves n (unlabel l-state # states)
  shows valid-labeled-moves-max-stone n (label-moves-max-stone l-state states)
  using assms

```

```

proof (induction states arbitrary: l-state)
  case Nil
  then show ?case
    by (simp add: valid-labeled-moves-max-stone-def)

```

```

next
  case (Cons state' states)
  let ?state = unlabel l-state
  let ?p = move-positions ?state state'
  let ?p1 = fst ?p
  let ?p2 = snd ?p
  let ?stone = Max (l-state ! ?p1)
  let ?l-state' = labeled-move ?p1 ?p2 ?stone l-state

```

```

  have valid-state n ?state
    using ⟨valid-labeled-state n l-state⟩
    by (simp add: unlabel-valid)

```

```

  have valid-move n ?state state'
    using Cons(3)
    by (metis Groups.add-ac(2) One-nat-def add-diff-cancel-left' add-is-0 grOI
list.size(4) n-not-Suc-n nth-Cons-0 nth-Cons-Suc plus-1-eq-Suc valid-moves-def)

```

```

  have valid-move' n ?p1 ?p2 ?state state'
    using ⟨valid-state n ?state⟩ ⟨valid-move n ?state state'⟩
    by (simp add: move-positions-valid-move')

```

```

have **: valid-labeled-move' n ?p1 ?p2 ?stone l-state ?l-state'
proof (rule valid-labeled-move-move-max-stone)
  show valid-labeled-state n l-state
    by fact
next
  show unlabel l-state = unlabel l-state
    by simp
next
  show valid-move' n ?p1 ?p2 ?state state'
    by fact
qed simp

have move ?p1 ?p2 ?state = state'
  using ⟨valid-move' n ?p1 ?p2 ?state state'⟩
  unfolding valid-move'-def Let-def
  by simp
then have *: unlabel ?l-state' = state'
  using unlabel-valid-move'[OF Cons(2) **, THEN conjunct2]
  by simp

have ***: valid-labeled-move-max-stone n l-state ?l-state'
  using **
  unfolding valid-labeled-move-max-stone-def
  by blast

have valid-labeled-moves-max-stone n (label-moves-max-stone ?l-state' states)
proof (rule Cons(1))
  show valid-moves n ((unlabel ?l-state') # states)
    using Cons(3) *
    using less-diff-conv valid-moves-def
    by auto
  have valid-labeled-move n l-state ?l-state'
    using **
    unfolding valid-labeled-move-def
    by metis
  then show valid-labeled-state n ?l-state'
    using Cons(2)
    using valid-labeled-move-valid-labeled-state
    by blast
qed

```

```

moreover
have  $hd\ (label-moves-max-stone\ ?l-state'\ states) = ?l-state'$ 
  using  $hd-label-moves-max-stone$  by  $blast$ 
ultimately
show  $?case$ 
  using  $***\ \langle valid-move'\ n\ ?p1\ ?p2\ ?state\ state'\ \rangle\ \langle valid-state\ n\ ?state\rangle$ 
  using  $valid-labeled-moves-max-stone-Cons$ 
  by  $(metis\ (mono-tags,\ lifting)\ case-prod-conv\ label-moves-max-stone.simps(2)\$ 
 $valid-move'-move-positions)$ 
qed

lemma  $final-labeled-state-unique$ :
  assumes  $unlabel\ l-state = final-state\ n\ valid-labeled-state\ n\ l-state$ 
  shows  $l-state = final-labeled-state\ n$ 
proof –
  have  $\forall\ i \leq n. finite\ (l-state\ !\ i)$ 
    by  $(metis\ Groups.add-ac(2)\ Union-upper\ assms(2)\ finite-atLeastLessThan$ 
 $finite-subset\ le-imp-less-Suc\ nth-mem\ plus-1-eq-Suc\ valid-labeled-state-def)$ 
  moreover
  have  $\forall\ i < n. card\ (l-state\ !\ i) = 0$ 
    using  $assms$ 
    unfolding  $unlabel-def\ final-state-def\ valid-labeled-state-def$ 
    by  $(metis\ One-nat-def\ add.right-neutral\ add-Suc-right\ le-imp-less-Suc\ less-imp-le-nat$ 
 $nat-neq-iff\ nth-list-update-neq\ nth-map\ nth-replicate)$ 
  moreover
  have  $card\ (l-state\ !\ n) = n$ 
    using  $assms$ 
    unfolding  $unlabel-def\ final-state-def\ valid-labeled-state-def$ 
    by  $(metis\ length-replicate\ less-add-same-cancel1\ less-one\ nth-list-update-eq\ nth-map)$ 
  moreover
  have  $\bigcup\ (set\ l-state) = \{0..<n\}\ length\ l-state = n + 1$ 
    using  $assms$ 
    unfolding  $unlabel-def\ final-state-def\ valid-labeled-state-def$ 
    by  $simp-all$ 
  ultimately
  have  $\forall\ i < n. l-state\ !\ i = \{\}\ l-state\ !\ n = \{0..<n\}$ 
    apply –
    apply  $auto[1]$ 
    apply  $(metis\ Union-upper\ assms(2)\ card-atLeastLessThan\ card-subset-eq\ diff-zero$ 
 $finite-atLeastLessThan\ less-add-same-cancel1\ nth-mem\ valid-labeled-state-def\ zero-less-one)$ 

```

```

done
show ?thesis
proof (rule nth-equalityI)
  show length l-state = length (final-labeled-state n)
    using ⟨length l-state = n + 1⟩
    unfolding final-labeled-state-def
    by (simp del: replicate-Suc)
next
fix i
assume i < length l-state
then show l-state ! i = final-labeled-state n ! i
  using ⟨∀ i < n. l-state ! i = { }⟩ ⟨l-state ! n = {0..<n}⟩ ⟨length l-state = n
+ 1⟩
  unfolding final-labeled-state-def
  by (metis add commute length-replicate less-Suc-eq nth-list-update-eq nth-list-update-neq
nth-replicate plus-1-eq-Suc)
qed
qed

```

lemma *labeled-game-max-stone-length* [simp]:

```

  assumes valid-game n states
  shows length (label-moves-max-stone (initial-labeled-state n) (tl states)) = length
states
  by (metis assms hd-Cons-tl length-map list.size(3) not-le unlabel-initial unlabel-label-moves-max-ston
valid-game-def valid-labeled-state-initial-labeled-state zero-less-numeral)

```

lemma *valid-labeled-game-max-stone*:

```

  assumes valid-game n states
  shows valid-labeled-game-max-stone n (label-moves-max-stone (initial-labeled-state
n) (tl states))
  unfolding valid-labeled-game-max-stone-def
proof safe
  let ?l-states = label-moves-max-stone (initial-labeled-state n) (tl states)
  have valid-moves n (unlabel (initial-labeled-state n) # tl states)
    using assms
  by (metis Groups.add-ac(2) One-nat-def add-diff-cancel-left' hd-Cons-tl list.sel(2)
list.size(3) list.size(4) n-not-Suc-n plus-1-eq-Suc unlabel-initial upt-0 upt-rec valid-game-def
valid-moves-def)
  then have *: map unlabel ?l-states = (initial-state n) # tl states
    using unlabel-label-moves-max-stone[of n initial-labeled-state n tl states]

```



```

  by simp

have unlabel (hd ?l-states) = initial-state n
  using *
  by auto
then show hd ?l-states = initial-labeled-state n
  by simp

have unlabel (last ?l-states) = final-state n
  using assms
  unfolding valid-game-def
  by (metis * Nil-is-map-conv hd-Cons-tl last-map list.size(3) not-le zero-less-numeral)
moreover
have valid-labeled-state n (last ?l-states)
  using * ⟨valid-moves n (unlabel (initial-labeled-state n) # tl states)⟩
  by (metis last-in-set list.discI list.simps(8) valid-labeled-state-initial-labeled-state
valid-states-label-moves-max-stone)
ultimately
show last ?l-states = final-labeled-state n
  using final-labeled-state-unique
  by blast

show valid-labeled-moves-max-stone n (label-moves-max-stone (initial-labeled-state
n) (tl states))
proof (rule label-moves-max-stone-valid-moves)
  show valid-labeled-state n (initial-labeled-state n)
    by simp
next
  show valid-moves n (unlabel (initial-labeled-state n) # tl states)
    by fact
qed
next
show  $2 \leq \text{length } (\text{label-moves-max-stone } (\text{initial-labeled-state } n) (\text{tl states}))$ 
  using assms
  unfolding valid-game-def
  by auto
qed

```

Valid labeled game move max stone length

lemma *moved-from:*

assumes *valid-labeled-state* n (*hd* l -states) *valid-labeled-moves* n l -states

$i < j$ $j < \text{length } l\text{-states}$ $\text{stone} < n$

$\text{stone-position } (l\text{-states} ! i) \text{ stone} \neq \text{stone-position } (l\text{-states} ! j) \text{ stone}$

shows $(\exists k. i \leq k \wedge k < j \wedge$

$(\text{let } (p1, p2, \text{stone}') = \text{labeled-move-positions } (l\text{-states} ! k) (l\text{-states} ! (k + 1)) \text{ in}$

$\text{stone}' = \text{stone} \wedge p1 = \text{stone-position } (l\text{-states} ! i) \text{ stone}))$

using *assms*

proof (*induction* l -states *arbitrary: i j*)

case *Nil*

then show ?*case*

by *simp*

next

case (*Cons* l -state l -states)

obtain $p1$ $p2$ stone' **where**

$*: (p1, p2, \text{stone}') = \text{labeled-move-positions } ((l\text{-state} \# l\text{-states}) ! i) ((l\text{-state} \# l\text{-states}) ! (i + 1))$

by (*metis prod-cases3*)

moreover

have ***: *valid-labeled-state* n $((l\text{-state} \# l\text{-states}) ! i)$

using *Cons(2-5)*

by (*meson less-imp-le-nat less-le-trans nth-mem valid-labeled-moves-valid-labeled-states*)

moreover

have *valid-labeled-move* n $((l\text{-state} \# l\text{-states}) ! i) ((l\text{-state} \# l\text{-states}) ! (i + 1))$

using *Cons(3-5)*

unfolding *valid-labeled-moves-def*

by *auto*

ultimately

have **: *valid-labeled-move'* n $p1$ $p2$ stone' $((l\text{-state} \# l\text{-states}) ! i) ((l\text{-state} \# l\text{-states}) ! (i + 1))$

using *labeled-move-positions-valid-move'*

```

by simp

show ?case
proof (cases stone' = stone)
  case True
  have p1 = stone-position ((l-state # l-states) ! i) stone'
  using **
  using (valid-labeled-state n ((l-state # l-states) ! i)) valid-labeled-move'-stone-positions
  by blast
  then show ?thesis
  using * Cons(4) True
  by (rule-tac x=i in exI, auto)
next
case False

  have stone-position ((l-state # l-states) ! (i + 1)) stone = stone-position
  ((l-state # l-states) ! i) stone
  using valid-labeled-move'-stone-positions-other[OF *** **] (stone' ≠ stone)
  (stone < n)
  by auto
  then have *: stone-position (l-states ! i) stone ≠ stone-position (l-states ! (j
  - 1)) stone
  using Cons(4) Cons(7)
  by auto
  moreover
  have valid-labeled-state n (hd l-states)
  proof-
  have l-states ≠ []
  using Cons(4) Cons(5) *
  by auto
  then show ?thesis
  using Cons(2-3)
  by (meson hd-in-set list.set-intros(2) valid-labeled-moves-valid-labeled-states)
qed

moreover
have valid-labeled-moves n l-states
  using Cons(3)
  using Groups.add-ac(2) less-diff-conv valid-labeled-moves-def
  by auto

```

```

moreover
have  $i < j - 1$ 
  using  $\text{Cons}(4) *$ 
  using  $\text{less-antisym plus-1-eq-Suc}$ 
  by  $\text{fastforce}$ 
moreover
have  $j - 1 < \text{length } l\text{-states}$ 
  using  $\langle i < j \rangle \text{Cons}(5)$ 
  by  $\text{auto}$ 
ultimately
obtain  $k$  where  $i \leq k < j - 1$ 
   $\text{let } (p1, p2, \text{stone}') = \text{labeled-move-positions } (l\text{-states} ! k) (l\text{-states} ! (k + 1)) \text{ in}$ 
     $\text{stone}' = \text{stone} \wedge p1 = \text{stone-position } (l\text{-states} ! i) \text{ stone}$ 
    using  $\text{Cons}(1)[\text{of } i \ j-1] \langle \text{stone} < n \rangle$ 
    by  $(\text{auto simp add: nth-Cons})$ 
    then show  $?thesis$ 
      using  $\langle \text{stone-position } ((l\text{-state} \# l\text{-states}) ! (i + 1)) \text{ stone} = \text{stone-position } ((l\text{-state} \# l\text{-states}) ! i) \text{ stone} \rangle$ 
      by  $(\text{rule-tac } x=k+1 \text{ in } exI) (\text{auto simp add: Let-def nth-Cons})$ 
qed
qed

```

lemma *valid-labeled-game-max-stone-min-length:*

assumes *valid-labeled-game-max-stone* n $l\text{-states}$

shows $\text{length } l\text{-states} \geq (\sum k \leftarrow [1..<n+1]. (\text{ceiling } (n / k))) + 1$

using *assms*

proof–

have $l\text{-states} \neq []$ $\text{length } l\text{-states} \geq 2$ *valid-labeled-state* n $(\text{hd } l\text{-states})$ *valid-labeled-moves* n $l\text{-states}$

using *assms*

using *valid-labeled-game-max-stone-def*

using *valid-labeled-game-def valid-labeled-game-max-stone-valid-labeled-game*

by *auto*

let $?ss = \text{map } (\lambda (state, state'). (state, \text{labeled-move-positions } state \ state')) (\text{zip } (\text{butlast } l\text{-states}) (\text{tl } l\text{-states}))$

let $?sstone = \lambda \text{stone}. \text{filter } (\lambda (state, p1, p2, \text{stone}'). \text{stone}' = \text{stone}) \ ?ss$

have $(\sum k \leftarrow [1..<n+1]. (\text{ceiling } (n / k))) =$

```

      ( $\sum$  stone  $\leftarrow$   $[0..<n]$ . ( $\text{ceiling } (n / (\text{stone} + 1))$ )))
proof–
  have map ( $\lambda x. x + 1$ )  $[0..<n] = [1..<n+1]$ 
    using map-add-upt by blast
  then show ?thesis
    by (subst sum-list-comp, simp)
qed
also have ...  $\leq (\sum$  stone  $\leftarrow$   $[0..<n]$ .  $\text{int } (\text{length } (?sstone \text{ stone}))$ )
proof (rule sum-list-mono)
  fix stone
  assume stone  $\in \text{set } [0..<n]$ 
  show  $\text{ceiling } (n / (\text{stone} + 1)) \leq \text{int } (\text{length } (?sstone \text{ stone}))$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then have  $\text{ceiling } (n / (\text{stone} + 1)) > \text{int } (\text{length } (?sstone \text{ stone}))$ 
      by simp
    then have  $\text{int } (\text{length } (?sstone \text{ stone})) * (\text{stone} + 1) < n$ 
      using lt-ceiling-frac
      by simp
    then have  $\text{length } (?sstone \text{ stone}) * (\text{stone} + 1) < n$ 
      by (metis (mono-tags, lifting) of-nat-less-imp-less of-nat-mult)

  obtain ss where ss: ss = ?sstone stone
    by auto

  have valid-moves':  $\forall (state, p1, p2, stone') \in \text{set } ss. stone' = \text{Max } (state !$ 
  p1)  $\wedge (\exists state'. \text{valid-labeled-move}' n p1 p2 stone' state state')$ 
  proof safe
    fix state p1 p2 stone'
    assume (state, p1, p2, stone')  $\in \text{set } ss$ 
    then have (state, p1, p2, stone')  $\in \text{set } ?ss$ 
      using ss
      by auto
    then obtain state' where
      (state, p1, p2, stone') = (state, labeled-move-positions state state')
      (state, state')  $\in \text{set } (\text{zip } (\text{butlast } l\text{-states}) (\text{tl } l\text{-states}))$ 
      by auto
    then obtain i where i < length ((zip (butlast l-states) (tl l-states))) (zip
    (butlast l-states) (tl l-states)) ! i = (state, state')
      by (meson in-set-conv-nth)

```

```

then have  $i < \text{length } l\text{-states} - 1$   $l\text{-states} ! i = \text{state } l\text{-states} ! (i + 1) =$ 
 $\text{state}'$ 
  using  $\text{nth-butlast}[of\ i\ l\text{-states}]\ \text{nth-tl}[of\ i\ l\text{-states}]$ 
  by simp-all
then have  $\text{valid-labeled-move-max-stone } n\ \text{state } \text{state}'$ 
  using  $\langle \text{valid-labeled-game-max-stone } n\ l\text{-states} \rangle$ 
unfolding  $\text{valid-labeled-game-max-stone-def } \text{valid-labeled-moves-max-stone-def}$ 
  by auto
moreover
  have  $\text{valid-labeled-state } n\ \text{state}$ 
    using  $\langle i < \text{length } l\text{-states} - 1 \rangle \langle l\text{-states} ! i = \text{state} \rangle$ 
  by  $(\text{meson } \text{add-lessD1 } \text{assms}(1) \text{ less-diff-conv } \text{nth-mem } \text{valid-labeled-game-max-stone-valid-label}$ 
 $\text{valid-labeled-game-valid-labeled-states})$ 
  ultimately
  have  $*$ :  $\text{valid-labeled-move}'\ n\ p1\ p2\ (\text{Max } (\text{state} ! p1))\ \text{state } \text{state}'$ 
    using  $\text{labeled-move-positions } \text{valid-labeled-move-max-stone-def}$ 
    using  $\langle (\text{state}, p1, p2, \text{stone}') = (\text{state}, \text{labeled-move-positions } \text{state } \text{state}') \rangle$ 
    by auto

  show  $\text{stone}' = \text{Max } (\text{state} ! p1)$ 
    using  $\langle (\text{state}, p1, p2, \text{stone}') = (\text{state}, \text{labeled-move-positions } \text{state } \text{state}') \rangle$ 
 $\langle \text{valid-labeled-move}'\ n\ p1\ p2\ (\text{Max } (\text{state} ! p1))\ \text{state } \text{state}' \rangle$ 
 $\langle \text{valid-labeled-state } n\ \text{state} \rangle$ 
     $\text{labeled-move-positions}$  by auto

  then show  $(\exists\ \text{state}'.\ \text{valid-labeled-move}'\ n\ p1\ p2\ \text{stone}'\ \text{state } \text{state}')$ 
    using  $*$ 
    by blast
qed

have  $\text{pos0}: \text{stone-position } (l\text{-states} ! 0)\ \text{stone} = 0$ 
  using  $\langle \text{stone} \in \text{set } [0..<n] \rangle \langle l\text{-states} \neq [] \rangle$ 
  using  $\langle \text{valid-labeled-game-max-stone } n\ l\text{-states} \rangle$ 
  using  $\text{stone-positionI}[of\ n\ l\text{-states} ! 0\ \text{stone } 0]$ 
  using  $\text{hd-conv-nth}[of\ l\text{-states},\ \text{symmetric}]$ 
  using  $\text{valid-labeled-state-initial-labeled-state}$ 
  unfolding  $\text{valid-labeled-game-max-stone-def } \text{initial-labeled-state-def}$ 
  by auto

have  $\text{posn}: \text{stone-position } (l\text{-states} ! (\text{length } l\text{-states} - 1))\ \text{stone} = n$ 
  using  $\text{stone-positionI}[of\ n\ l\text{-states} ! (\text{length } l\text{-states} - 1)\ \text{stone } n]$ 

```

```

using  $\langle \text{stone} \in \text{set } [0..<n] \rangle \langle l\text{-states} \neq [] \rangle$ 
using  $\langle \text{valid-labeled-game-max-stone } n \text{ } l\text{-states} \rangle$ 
using  $\text{last-conv-nth}[of \text{ } l\text{-states}, \text{ symmetric}]$ 
using  $\text{valid-labeled-state-final-labeled-state}$ 
unfolding  $\text{valid-labeled-game-max-stone-def final-labeled-state-def}$ 
by  $(\text{simp del: replicate-Suc})$ 

have  $n > 0$ 
  using  $\langle \text{length } (?sstone \text{ stone}) * (\text{stone} + 1) < n \rangle \text{ gr-implies-not0}$ 
  by  $\text{blast}$ 

have  $\text{length } ss \geq 1$ 
proof  $(\text{rule ccontr})$ 
  assume  $\neg ?thesis$ 
  then have  $?sstone \text{ stone} = []$ 
    using  $ss$ 
    by  $(\text{metis One-nat-def Suc-leI length-greater-0-conv})$ 

have  $\text{valid-labeled-moves } n \text{ } l\text{-states}$ 
  using  $\langle \text{valid-labeled-game-max-stone } n \text{ } l\text{-states} \rangle$ 
  unfolding  $\text{valid-labeled-game-max-stone-def}$ 
using  $\text{assms valid-labeled-game-def valid-labeled-game-max-stone-valid-labeled-game}$ 
  by  $\text{blast}$ 

then obtain  $p2 \text{ } k$  where  $k < \text{length } l\text{-states} - 1$ 
   $(0, p2, \text{stone}) = \text{labeled-move-positions } (l\text{-states} ! k) (l\text{-states} ! (k + 1))$ 
  using  $\text{moved-from}[of \text{ } n \text{ } l\text{-states } 0 \text{ length } l\text{-states} - 1 \text{ stone}]$ 
  using  $\text{pos0 posn } \langle n > 0 \rangle \langle \text{stone} \in \text{set } [0..<n] \rangle$ 
  using  $\langle \text{valid-labeled-game-max-stone } n \text{ } l\text{-states} \rangle$ 
  unfolding  $\text{valid-labeled-game-max-stone-def}$ 
  by  $\text{force}$ 
moreover
have  $(l\text{-states} ! k, l\text{-states} ! (k+1)) \in \text{set } (\text{zip } (\text{butlast } l\text{-states}) (\text{tl } l\text{-states}))$ 
  using  $\langle k < \text{length } l\text{-states} - 1 \rangle \langle \text{length } l\text{-states} \geq 2 \rangle$ 
  by  $(\text{metis } (\text{no-types}, \text{lifting}) \text{ One-nat-def add.right-neutral add-Suc-right}$ 
 $\text{in-set-conv-nth length-butlast length-tl length-zip min-less-iff-conj nth-butlast nth-tl}$ 
 $\text{nth-zip})$ 
ultimately
have  $(l\text{-states} ! k, 0, p2, \text{stone}) \in \text{set } (?sstone \text{ stone})$ 
  by  $\text{auto}$ 

```

```

then show False
  using ⟨?sstone stone = []⟩
  by auto
qed
then have ss ≠ []
  by auto

have n = (∑ (state, p1, p2, stone) ← ?sstone stone. p2 - p1)
proof-
  let ?p2p1 = λ i. case ss ! i of (state, p1, p2, stone) ⇒ int p2 - int p1
  let ?p1 = λ i. case ss ! i of (state, p1, p2, stone) ⇒ int p1
  let ?p2 = λ i. case ss ! i of (state, p1, p2, stone) ⇒ int p2

  have (∑ (state, p1, p2, stone) ← ss. p2 - p1) =
    (∑ (state, p1, p2, stone) ← ss. int (p2 - p1))
  proof-
    have (∑ (state, p1, p2, stone) ← ss. p2 - p1) =
      (∑ x ← map (λ (state, p1, p2, stone). p2 - p1) ss. int x)
    by (metis (no-types) map-nth sum-list-comp sum-list-int)
    also have ... = (∑ (state, p1, p2, stone) ← ss. int (p2 - p1))
    proof-
      have *: (map int (map (λ (state, p1, p2, stone). p2 - p1) ss)) =
        (map (λ (state, p1, p2, stone). int (p2 - p1)) ss)
      by auto
      show ?thesis
      by (subst *, simp)
    qed
    finally show ?thesis
  .
qed
also have ... = (∑ (state, p1, p2, stone) ← ss. int p2 - int p1)
proof-
  have ∀ (state, p1, p2, stone) ∈ set ss. p2 ≥ p1
  using valid-moves'
  unfolding valid-labeled-move'-def Let-def
  by auto
  then have ∀ (state, p1, p2, stone) ∈ set ss. int (p2 - p1) = int p2 -
int p1
  by auto
  then have *: map (λ (state, p1, p2, stone). int (p2 - p1)) ss =

```



```

      map (λ (state, p1, p2, stone). int p2 - int p1) ss
    by auto
  show ?thesis
    by (subst *, simp)
qed
also have ... = (∑ i ← [0..

```

```

let ?P =  $\lambda(state, p1, p2, stone'). stone' = stone$ 

have ( $\sum i \leftarrow [1..<length\ ss]. ?p1\ i$ ) = ( $\sum i \leftarrow [0..<length\ ss - 1]. ?p2\ i$ )
proof–
  have *:  $\forall i. 0 < i \wedge i < length\ ss \longrightarrow ?p1\ i = ?p2\ (i-1)$ 
  proof safe
    fix i
    assume  $0 < i \wedge i < length\ ss$ 
    show  $?p1\ i = ?p2\ (i-1)$ 
    proof (rule ccontr)
      assume  $\neg ?thesis$ 
      obtain k1 k2 where
         $k1 < k2 \wedge k2 < length\ ?ss$ 
         $ss ! (i-1) = ?ss ! k1 \wedge ss ! i = ?ss ! k2$ 
         $?P\ (?ss ! k1)\ ?P\ (?ss ! k2) \wedge k'. k1 < k' \wedge k' < k2 \longrightarrow \neg ?P\ (?ss$ 
! k')

      using ss inside-filter[of i-1 ?P ?ss] <0 < i> <i < length ss>
      using <ss ≠ []> <length l-states ≥ 2>
      by force
      have  $k2 < length\ l\text{-states}$ 
      using <k2 < length ?ss>
      by simp
      have  $?ss ! k1 = (l\text{-states} ! k1, labeled\text{-move-positions}\ (l\text{-states} ! k1)$ 
(l-states ! (k1+1)))
       $?ss ! k2 = (l\text{-states} ! k2, labeled\text{-move-positions}\ (l\text{-states} ! k2)$ 
(l-states ! (k2+1)))
      using <k1 < k2> <k2 < length ?ss> <length l-states ≥ 2>
      by (auto simp add: nth-butlast nth-tl)
      then obtain p1a p2a p1b p2b where
         $?ss ! k1 = (l\text{-states} ! k1, p1a, p2a, stone)\ labeled\text{-move-positions}$ 
(l-states ! k1) (l-states ! (k1+1)) = (p1a, p2a, stone)
         $?ss ! k2 = (l\text{-states} ! k2, p1b, p2b, stone)\ labeled\text{-move-positions}$ 
(l-states ! k2) (l-states ! (k2+1)) = (p1b, p2b, stone)
      using <?P (?ss ! k1)> <?P (?ss ! k2)>
      by auto
      then have  $p2a \neq p1b$ 
      using <?p1 i ≠ ?p2 (i-1)> <ss ! (i-1) = ?ss ! k1> <ss ! i = ?ss ! k2>
      by simp

have stone-position (l-states ! (k1 + 1)) stone ≠ stone-position (l-states

```

```

! k2) stone
  proof-
    have valid-labeled-state n (l-states ! k1)
      by (meson <k1 < k2> <k2 < length l-states> assms less-imp-le-nat
less-le-trans nth-mem valid-labeled-game-max-stone-valid-labeled-game valid-labeled-game-valid-labeled-states)
    moreover
      then have valid-labeled-move' n p1a p2a stone (l-states ! k1) (l-states
! (k1+1))
        using <labeled-move-positions (l-states ! k1) (l-states ! (k1+1)) =
(p1a, p2a, stone)>
        using labeled-move-positions-valid-move'
        using <k1 < k2> <k2 < length l-states> <valid-labeled-moves n l-states>
valid-labeled-moves-def
        by auto
      ultimately
        have stone-position (l-states ! (k1 + 1)) stone = p2a
          using valid-labeled-move'-stone-positions
          by blast

    have valid-labeled-state n (l-states ! k2)
      by (meson <k2 < length l-states> assms less-imp-le-nat less-le-trans
nth-mem valid-labeled-game-max-stone-valid-labeled-game valid-labeled-game-valid-labeled-states)
    moreover
      then have valid-labeled-move' n p1b p2b stone (l-states ! k2) (l-states
! (k2+1))
        using <labeled-move-positions (l-states ! k2) (l-states ! (k2+1)) =
(p1b, p2b, stone)>
        using labeled-move-positions-valid-move'
        using <k2 < length ?ss> <valid-labeled-moves n l-states>
valid-labeled-moves-def
        by (smt length-butlast length-map length-tl length-zip min-less-iff-conj)
      ultimately
        have stone-position (l-states ! k2) stone = p1b
          using valid-labeled-move'-stone-positions
          by blast

    show ?thesis
      using <stone-position (l-states ! k2) stone = p1b>
      using <stone-position (l-states ! (k1 + 1)) stone = p2a>
      using <p2a ≠ p1b>

```

```

      by simp
    qed
  then have  $k1 + 1 < k2$ 
    using  $\langle k1 < k2 \rangle$ 
    by (metis Suc-eq-plus1 Suc-leI nat-less-le)
  then obtain  $k' p1'' p2''$  where  $k1 + 1 \leq k' k' < k2$ 
    ( $p1'', p2'', stone$ ) = labeled-move-positions ( $l\text{-states} ! k'$ ) ( $l\text{-states} !$ 
( $k' + 1$ ))
    using  $\langle stone\text{-position} (l\text{-states} ! (k1 + 1)) stone \neq stone\text{-position}$ 
( $l\text{-states} ! k2$ )  $stone \rangle$ 
    using moved-from[of  $n$   $l\text{-states}$   $k1+1$   $k2$   $stone$ ]  $\langle stone \in set [0..<n] \rangle$ 
    using  $\langle length\ l\text{-states} \geq 2 \rangle \langle k1 < k2 \rangle \langle k2 < length\ l\text{-states} \rangle$ 
    using  $\langle valid\text{-labeled-moves } n\ l\text{-states} \rangle \langle valid\text{-labeled-state } n\ (hd\ l\text{-states}) \rangle$ 
    by auto
  then have  $?ss ! k' = (l\text{-states} ! k', p1'', p2'', stone)$ 
    using  $\langle k2 < length\ ?ss \rangle$ 
    by (auto simp add: nth-butlast nth-tl)
  then show False
    using  $\langle \forall k'. k1 < k' \wedge k' < k2 \longrightarrow \neg ?P (?ss ! k') \rangle$  [rule-format, of
 $k'$ ]  $\langle k1 + 1 \leq k' \rangle \langle k' < k2 \rangle$ 
    by simp
  qed
qed
have map  $?p1 [1..<length\ ss] = map\ ?p2 [0..<length\ ss - 1]$  (is  $?lhs =$ 
 $?rhs$ )
proof (rule nth-equalityI)
  show  $length\ ?lhs = length\ ?rhs$ 
    by simp
next
  fix  $i$ 
  assume  $i < length\ ?lhs$ 
  then show  $?lhs ! i = ?rhs ! i$ 
    using *
    by simp
qed
then show  $?thesis$ 
  by simp
qed
moreover
have  $?p2 (length\ ss - 1) = n$ 

```

```

proof (rule ccontr)
  assume  $\neg ?thesis$ 
  obtain  $k$  where
     $k < \text{length } ?ss \text{ ss} ! (\text{length } ss - 1) = ?ss ! k \text{ ?P } (?ss ! k) \forall k'. k < k' \wedge k' < \text{length } ?ss \longrightarrow \neg ?P (?ss ! k')$ 
    using  $ss \text{ last-filter[of ?P ?ss]}$ 
    using  $\langle ss \neq [] \rangle \langle \text{length } l\text{-states} \geq 2 \rangle$ 
    by auto
  have  $k < \text{length } l\text{-states} - 1$ 
    using  $\langle k < \text{length } ?ss \rangle$ 
    by simp
  have  $?ss ! k = (l\text{-states} ! k, \text{labeled-move-positions } (l\text{-states} ! k) (l\text{-states} ! (k+1)))$ 
    using  $\langle k < \text{length } ?ss \rangle \langle \text{length } l\text{-states} \geq 2 \rangle$ 
    by (auto simp add: nth-butlast nth-tl)
  then obtain  $p1' p2'$  where  $?ss ! k = (l\text{-states} ! k, p1', p2', \text{stone})$ 
     $\text{labeled-move-positions } (l\text{-states} ! k) (l\text{-states} ! (k+1)) = (p1', p2', \text{stone})$ 
    using  $\langle ?P (?ss ! k) \rangle$ 
    by auto
  then have  $p2' \neq n$ 
    using  $\langle ?p2 (\text{length } ss - 1) \neq n \rangle \langle ss ! (\text{length } ss - 1) = ?ss ! k \rangle$ 
    by auto
  have  $\text{stone-position } (l\text{-states} ! (k + 1)) \text{ stone} \neq n$ 
  proof–
    have  $\text{stone-position } (l\text{-states} ! (k + 1)) \text{ stone} = p2'$ 
    proof–
      have  $\text{valid-labeled-move}' n p1' p2' \text{stone } (l\text{-states} ! k) (l\text{-states} ! (k+1))$ 
        using  $\langle \text{labeled-move-positions } (l\text{-states} ! k) (l\text{-states} ! (k+1)) = (p1', p2', \text{stone}) \rangle$ 
        using  $\langle k < \text{length } l\text{-states} - 1 \rangle \langle \text{valid-labeled-moves } n l\text{-states} \rangle$ 
         $\langle \text{valid-labeled-state } n (\text{hd } l\text{-states}) \rangle$ 
        by (meson add-lessD1 labeled-move-positions-valid-move' less-diff-conv nth-mem valid-labeled-moves-def valid-labeled-moves-valid-labeled-states)
      moreover
        have  $\text{valid-labeled-state } n (l\text{-states} ! k)$ 
          using  $\langle k < \text{length } l\text{-states} - 1 \rangle \langle \text{valid-labeled-moves } n l\text{-states} \rangle$ 
           $\langle \text{valid-labeled-state } n (\text{hd } l\text{-states}) \rangle$ 
          using  $\text{valid-labeled-moves-valid-labeled-states}$ 
          by auto
        ultimately

```

```

    show ?thesis
    using valid-labeled-move'-stone-positions
    by blast
qed
then show ?thesis
    using ⟨p2' ≠ n⟩
    by simp
qed
then have k + 1 < length l-states - 1
    using posn ⟨k < length l-states - 1⟩
    by (smt Nat.le-diff-conv2 Nat.le-imp-diff-is-add Suc-leI add.right-neutral
add-Suc-right add-leD2 diff-diff-left nat-less-le one-add-one plus-1-eq-Suc)
    then obtain k' p1'' p2'' where k' ≥ k + 1 k' < length l-states - 1 (p1'',
p2'', stone) = labeled-move-positions (l-states ! k') (l-states ! (k' + 1))
    using moved-from[of n l-states k+1 length l-states - 1 stone]
    using posn ⟨stone-position (l-states ! (k+1)) stone ≠ n⟩ ⟨stone ∈ set
[0.. $n$ ⟩
    using ⟨length l-states ≥ 2⟩
    using ⟨valid-labeled-moves n l-states⟩ ⟨valid-labeled-state n (hd l-states)⟩
    by force
    then have ?ss ! k' = (l-states ! k', p1'', p2'', stone)
    by (simp add: nth-butlast nth-tl)
    then show False
    using ⟨∀ k'. k < k' ∧ k' < length ?ss ⟶ ¬ ?P (?ss ! k')⟩[rule-format,
of k'] ⟨k' ≥ k + 1⟩ ⟨k' < length l-states - 1⟩
    by auto
qed
moreover
have ?p1 0 = 0
proof (rule ccontr)
    assume ¬ ?thesis
    obtain k where
        k < length ?ss ss ! 0 = ?ss ! k ?P (?ss ! k) ∀ k' < k. ¬ ?P (?ss ! k')
    using ss hd-filter[of ?P ?ss]
    using ⟨ss ≠ []⟩ ⟨length l-states ≥ 2⟩
    by auto
    have k < length l-states - 1
    using ⟨k < length ?ss⟩
    by simp
    have ?ss ! k = (l-states ! k, labeled-move-positions (l-states ! k) (l-states

```

```

! (k+1)))
  using ⟨k < length ?ss⟩ ⟨length l-states ≥ 2⟩
  by (auto simp add: nth-butlast nth-tl)
  then obtain p1' p2' where ?ss ! k = (l-states ! k, p1', p2', stone)
labeled-move-positions (l-states ! k) (l-states ! (k+1)) = (p1', p2', stone)
  using ⟨?P (?ss ! k)⟩
  by auto
then have p1' ≠ 0
  using ⟨?p1 0 ≠ 0⟩ ⟨ss ! 0 = ?ss ! k⟩
  by auto
have stone-position (l-states ! k) stone ≠ 0
proof-
  have valid-labeled-state n (l-states ! k)
    by (meson ⟨k < length l-states - 1⟩ add-lessD1 assms less-diff-conv
nth-mem valid-labeled-game-max-stone-valid-labeled-game valid-labeled-game-valid-labeled-states)
  moreover
  then have valid-labeled-move' n p1' p2' stone (l-states ! k) (l-states !
(k+1))
    using ⟨labeled-move-positions (l-states ! k) (l-states ! (k+1)) = (p1',
p2', stone)⟩
    using ⟨k < length l-states - 1⟩ ⟨valid-labeled-moves n l-states⟩
labeled-move-positions-valid-move' valid-labeled-moves-def
    by blast
  ultimately
  have stone-position (l-states ! k) stone = p1'
    using valid-labeled-move'-stone-positions
    by blast
  then show ?thesis
    using ⟨p1' ≠ 0⟩
    by simp
qed
then have k > 0
  using pos0
  using neq0-conv
  by blast
have k < length l-states
  using ⟨k < length l-states - 1⟩ ⟨length l-states ≥ 2⟩
  by auto
  then obtain k' p2'' where k' < k labeled-move-positions (l-states ! k')
(l-states ! (k' + 1)) = (0, p2'', stone)

```

```

using moved-from[of n l-states 0 k stone] pos0  $\langle \text{stone-position } (l\text{-states } !$ 
k) stone  $\neq 0 \rangle$ 
using  $\langle \text{valid-labeled-state } n \text{ (hd } l\text{-states)} \rangle \langle \text{valid-labeled-moves } n \text{ } l\text{-states} \rangle$ 
 $\langle k > 0 \rangle \langle \text{stone} \in \text{set } [0..<n] \rangle$ 
by auto
then have  $?ss ! k' = (l\text{-states } ! k', 0, p2'', \text{stone})$ 
using  $\langle k' < k \rangle \langle k < \text{length } l\text{-states} - 1 \rangle$ 
using  $\langle k < \text{length } ?ss \rangle \langle \text{length } l\text{-states} \geq 2 \rangle$ 
by (auto simp add: nth-butlast nth-tl)
then show False
using  $\langle \forall k' < k. \neg ?P (?ss ! k') [\text{rule-format, of } k] \rangle \langle k' < k \rangle$ 
by simp
qed
ultimately
show ?thesis
using ss
by simp
qed
also have  $\dots \leq (\sum (\text{state}, p1, p2, \text{stone}) \leftarrow ?ss\text{stone } \text{stone}. \text{stone} + 1)$ 
proof (rule sum-list-mono)
fix  $x :: \text{labeled-state} \times \text{nat} \times \text{nat} \times \text{nat}$ 
obtain  $\text{state } p1 \text{ } p2 \text{ } \text{stone}'$  where  $x: x = (\text{state}, p1, p2, \text{stone}')$ 
by (cases x)
assume  $x \in \text{set } (?ss\text{stone } \text{stone})$ 
then have  $x \in \text{set } ss$ 
using ss
by auto
then obtain  $\text{state}'$  where  $\text{stone}' = \text{Max } (\text{state } ! p1) \text{ valid-labeled-move}' n$ 
 $p1 \text{ } p2 \text{ } (\text{Max } (\text{state } ! p1)) \text{ state } \text{state}'$ 
using  $x \text{ valid-moves}'$ 
by auto
then have  $p1 < p2 \text{ } p2 \leq p1 + \text{card } (\text{state } ! p1)$ 
unfolding valid-labeled-move'-def Let-def
by auto

moreover

have  $\text{card } (\text{state } ! p1) \leq \text{Max } (\text{state } ! p1) + 1$ 
by (rule card-Max)

```


ultimately

```

show (case  $x$  of ( $state, p1, p2, stone$ )  $\Rightarrow p2 - p1$ )  $\leq$ 
      (case  $x$  of ( $state, p1, p2, stone$ )  $\Rightarrow stone + 1$ )
using  $x \langle stone' = Max (state ! p1) \rangle$ 
by simp
qed
also have ... = ( $\sum x \leftarrow ?ss_{stone} stone. stone + 1$ )
proof-
  have  $map (\lambda (state, p1, p2, stone). stone + 1) (?ss_{stone} stone) =$ 
     $map (\lambda x. stone + 1) (?ss_{stone} stone)$ 
  by auto
  then show ?thesis
  by presburger
qed
also have ... =  $length (?ss_{stone} stone) * (stone + 1)$ 
  by (simp add: sum-list-triv)
finally
show False
  using  $\langle length (?ss_{stone} stone) * (stone + 1) < n \rangle$ 
  by simp
qed
qed
also have ...  $\leq length ?ss$ 
proof-
  let  $?ps = map (\lambda stone. (\lambda (state, p1, p2, stone'). stone' = stone)) [0..<n]$ 
  have  $\forall i j. i < j \wedge j < length ?ps \longrightarrow set (filter (?ps ! i) ?ss) \cap set (filter$ 
     $(?ps ! j) ?ss) = \{\}$ 
  by auto
  then have ( $\sum stone \leftarrow [0..<n]. length (?ss_{stone} stone)$ )  $\leq length ?ss$ 
  using sum-length-parts[of ?ps ?ss]
  by (auto simp add: comp-def split: prod.split)
  then show ?thesis
  by (subst sum-list-int, simp)
qed
finally
have ( $\sum k \leftarrow [1..<n+1]. (ceiling (n / k))) + 1 \leq length ?ss + 1$ 
  by simp
moreover
have  $length ?ss + 1 = length l-states$ 

```

```

    using ⟨l-states ≠ []⟩
    by simp
  ultimately
  show ?thesis
    by simp
qed

```

Valid game length

theorem *IMO2018SL-C3*:

assumes *valid-game n states*

shows $\text{length } \text{states} \geq (\sum k \leftarrow [1..<n+1]. (\text{ceiling } (n / k))) + 1$

proof–

let *?l-states = label-moves-max-stone (initial-labeled-state n) (tl states)*

have $\text{length } ?l\text{-states} = \text{length } \text{states}$

using *assms*

unfolding *valid-game-def*

by *auto*

moreover

have *valid-labeled-game-max-stone n ?l-states*

using *valid-labeled-game-max-stone[OF assms]*

by *simp*

ultimately

show *?thesis*

using *valid-labeled-game-max-stone-min-length[of n ?l-states]*

by *simp*

qed

end

5.2.4 IMO 2018 SL - C4

theory *IMO-2018-SL-C4-sol*

imports *Main HOL-Library.Permutation*

begin

definition *antipascal* :: $(\text{nat} \Rightarrow \text{nat} \Rightarrow \text{int}) \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**

antipascal f n $\longleftrightarrow (\forall r < n. \forall c \leq r. f\ r\ c = \text{abs } (f\ (r+1)\ c - f\ (r+1)\ (c+1)))$

definition *triangle* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow (*nat* \times *nat*) *set* **where**
triangle *r0* *c0* *n* = $\{(r, c) \mid r \leq c :: \text{nat}. r0 \leq r \wedge r < r0 + n \wedge c0 \leq c \wedge c \leq c0 + r - r0\}$

lemma *triangle-finite* [*simp*]:

shows *finite* (*triangle* *r0* *c0* *n*)

proof—

have *triangle* *r0* *c0* *n* $\subseteq \{0..<r0 + n\} \times \{0..<c0 + n\}$

unfolding *triangle-def*

by *auto*

then show *?thesis*

using *finite-atLeastLessThan infinite-super*

by *blast*

qed

lemma *triangle-card*:

shows *card* (*triangle* *r0* *c0* *n*) = *n* * (*n*+1) *div* 2

proof (*induction* *n* *arbitrary*: *r0* *c0*)

case 0

have *: $\{(i, j). r0 \leq i \wedge i < r0 \wedge c0 \leq j \wedge j \leq c0 + i - r0\} = \{\}$

by *auto*

then show *?case*

using 0

unfolding *triangle-def*

by (*simp add*: *)

next

case (*Suc* *n*)

let *?row* = $\{(r0 + n, j) \mid j. c0 \leq j \wedge j < c0 + \text{Suc } n\}$

have *triangle* *r0* *c0* (*Suc* *n*) = *triangle* *r0* *c0* *n* \cup *?row*

unfolding *triangle-def*

by *auto*

moreover

have *triangle* *r0* *c0* *n* \cap *?row* = $\{\}$

unfolding *triangle-def*

by *auto*

ultimately

have *card* (*triangle* *r0* *c0* (*Suc* *n*)) = *card* (*triangle* *r0* *c0* *n*) + *card* *?row*

by (*simp add*: *card-Un-disjoint*)

moreover

have *card* *?row* = *Suc* *n*

```

proof–
  have ?row = ( $\lambda j. (r0 + n, j)$ ) ‘ {c0.. $c0 + Suc\ n$ }
    by auto
  moreover
  have inj-on ( $\lambda j. (r0 + n, j)$ ) {c0.. $c0 + Suc\ n$ }
    unfolding inj-on-def
    by auto
  then have  $card\ ((\lambda j. (r0 + n, j))\ ‘\ \{c0.. $c0 + Suc\ n\}) = card\ \{c0.. $c0 + Suc\ n\}$$ 
    using card-image
    by blast
  then have  $card\ ((\lambda j. (r0 + n, j))\ ‘\ \{c0.. $c0 + Suc\ n\}) = Suc\ n$ 
    by auto
  ultimately
  show ?thesis
    by simp
qed
ultimately
have  $card\ (triangle\ r0\ c0\ (Suc\ n)) = (n * (n + 1))\ div\ 2 + Suc\ n$ 
  using Suc
  by simp
then show ?case
  by auto
qed$$ 
```

```

fun uncurry where
  uncurry  $f\ (a,\ b) = f\ a\ b$ 

```

```

lemma gauss:
  fixes  $n :: nat$ 
  shows  $sum-list\ [1.. $n]$  =  $n * (n - 1)\ div\ 2$ 
proof (induction  $n$ )
  case 0
  then show ?case by simp
next
  case ( $Suc\ n$ )
  have  $sum-list\ [1.. $Suc\ n]$  =  $sum-list\ [1.. $n]$  +  $n$$ 
    by simp
  also have  $\dots = n * (n - 1)\ div\ 2 + n$ 
    using Suc$$ 
```

```

    by simp
  finally
  show ?case
    by (metis Sum-Ico-nat diff-self-eq-0 distinct-sum-list-conv-Sum distinct-upt
      minus-nat.diff-0 mult-eq-0-iff set-upt)
qed

```

```

lemma sum-list-insort [simp]:
  fixes x :: nat and xs :: nat list
  shows sum-list (insort x xs) = x + sum-list xs
  by (induction xs, auto)

```

```

lemma sum-list-sort [simp]:
  fixes xs :: nat list
  shows sum-list (sort xs) = sum-list xs
  by (induction xs, auto)

```

```

lemma sorted-distinct-strict-increase:
  assumes sorted (xs @ [x]) distinct (xs @ [x])  $\forall x \in \text{set } (xs @ [x]). x > a$ 
  shows  $x > a + \text{length } xs$ 
  using assms
proof (induction xs arbitrary: x rule: rev-induct)
  case Nil
  then show ?case
    by simp
next
  case (snoc x' xs)
  show ?case
    using snoc(1)[of x'] snoc(2-)
    by (auto simp add: sorted-append)
qed

```

```

lemma sum-list-sorted-distinct-lb:
  assumes  $\forall x \in \text{set } xs. x > a$  distinct xs sorted xs
  shows  $\text{sum-list } xs \geq \text{length } xs * (2 * a + \text{length } xs + 1) \text{ div } 2$ 
  using assms
proof (induction xs rule: rev-induct)
  case Nil
  then show ?case
    by simp

```

```

next
  case (snoc x xs)
  have  $x > a + \text{length } xs$ 
    using sorted-distinct-strict-increase[of xs x a]
    using snoc(2-)
    by auto
  moreover
  have  $\text{length } xs * (2 * a + \text{length } xs + 1) \text{ div } 2 \leq \text{sum-list } xs$ 
    using snoc
    by (auto simp add: sorted-append)
  ultimately
  show ?case
    by auto
qed

```

```

lemma sum-list-distinct-lb:
  assumes  $\forall x \in \text{set } xs. f x > a \text{ distinct } (\text{map } f xs)$ 
  shows  $(\sum x \leftarrow xs. f x) \geq \text{length } xs * (2 * a + \text{length } xs + 1) \text{ div } 2$ 
  using assms
  using sum-list-sorted-distinct-lb[of sort (map f xs) a]
  by simp

```

```

lemma consecutive-nats-sorted:
  assumes sorted xs length xs = n distinct xs sum-list xs  $\leq n * (n + 1) \text{ div } 2 \forall$ 
 $x \in \text{set } xs. x > 0$ 
  shows  $xs = [1..<n+1]$ 
  using assms
proof (induction xs arbitrary: n rule: rev-induct)
  case Nil
  then show ?case
    by simp
next
  case (snoc x xs)
  have  $n > 0$ 
    using  $\langle \text{length } (xs @ [x]) = n \rangle$ 
    by simp
  have  $xs = [1..<(n-1)+1]$ 
  proof (rule snoc(1))
    show sorted xs length xs = n-1 distinct xs  $\forall a \in \text{set } xs. 0 < a$ 
      using snoc(2-6)

```

```

  by (auto simp add: sorted-append)
show sum-list xs ≤ (n − 1) * (n − 1 + 1) div 2
proof −
  have x ≥ n
    using snoc(2-4) snoc(6)
  proof (induction xs arbitrary: x n rule: rev-induct)
    case Nil
    then show ?case
      by simp
  next
    case (snoc x' xs')
    have n − 1 ≤ x'
      using snoc(1)[of x' n − 1] snoc(2 −)
      by (simp add: sorted-append)
    moreover
    have x > x'
      using snoc(2) snoc(4)
      by (simp add: sorted-append)
    ultimately
    show ?case
      by simp
  qed
show ?thesis
proof −
  have sum-list xs ≤ n * (n + 1) div 2 − x
    using snoc(5)
    by simp
  also have ... ≤ n * (n + 1) div 2 − n
    using ⟨n ≤ x⟩
    by simp
  also have ... = n * (n − 1) div 2
    by (simp add: diff-mult-distrib2)
  finally
  show ?thesis
    using ⟨n > 0⟩
    by (auto simp add: mult.commute)
  qed
qed
qed
then have xs = [1..n]

```

```

    using ⟨ $n > 0$ ⟩
    by simp
  then have  $x \geq n$ 
    using snoc(2) snoc(4) snoc(6)
    by (auto simp add: sorted-append)
  have  $x = n$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then have  $x > n$ 
      using ⟨ $x \geq n$ ⟩
      by simp
    then have  $\text{sum-list } (xs @ [x]) > n * (n - 1) \text{ div } 2 + n$ 
      using ⟨ $xs = [1..<n]$ ⟩ gauss[of  $n$ ]
      by simp
    then show False
      using snoc(5)
      by (smt Suc-diff-1 ⟨ $0 < n$ ⟩ add.commute add-Suc-right distrib-left div-mult-self2
        less-le-trans mult-2 mult-2-right nat-neq-iff one-add-one plus-1-eq-Suc zero-neq-numeral)
    qed
  then show ?case
    using ⟨ $xs = [1..<n]$ ⟩
    by (simp add: Suc-leI ⟨ $0 < n$ ⟩)
  qed

```

lemma *consecutive-nats*:

```

  assumes  $\text{length } xs = n \text{ distinct } xs \text{ sum-list } xs \leq n * (n + 1) \text{ div } 2 \ \forall \ x \in \text{set } xs. \ x > 0$ 
  shows  $\text{set } xs = \{1..<n+1\}$ 
  proof-
    have  $\text{sort } xs = [1..<n+1]$ 
      using consecutive-nats-sorted[of  $\text{sort } xs \ n$ ] assms
      by simp
    then show ?thesis
      by (metis set-sort set-upt)
  qed

```

lemma *sum-list-cong*:

```

  assumes  $\forall \ x \in \text{set } xs. \ f \ x = g \ x$ 
  shows  $(\sum \ x \leftarrow xs. \ f \ x) = (\sum \ x \leftarrow xs. \ g \ x)$ 
  using assms

```


by (*induction xs, auto*)

lemma *sum-list-last*:

assumes $a \leq b$

shows $(\sum x \leftarrow [a..<b+1]. f\ x) = (\sum x \leftarrow [a..<b]. f\ x) + f\ b$

proof–

have *: $[a..<b+1] = [a..<b] @ [b]$

using *assms*

by *auto*

show *?thesis*

by (*subst *, simp*)

qed

lemma *sum-list-nat*:

assumes $\forall x \in \text{set } xs. f\ x \geq 0$

shows $(\sum x \leftarrow xs. \text{nat } (f\ x)) = (\text{nat } (\sum x \leftarrow xs. f\ x))$

using *assms*

proof (*induction xs*)

case *Nil*

then show *?case*

by *simp*

next

case (*Cons x xs*)

then show *?case*

using *sum-list-mono*

by *fastforce*

qed

theorem *IMO2018SL-C4*:

$\nexists f. \text{antipascal } f\ 2018 \wedge$

$(\text{uncurry } f)\ ' \text{triangle } 0\ 0\ 2018 = \{1..<2018*(2018 + 1) \text{ div } 2 + 1\}$

proof (*rule ccontr*)

assume $\neg ?thesis$

then obtain *f* where

$f: \text{antipascal } f\ 2018\ (\text{uncurry } f)\ ' \text{triangle } 0\ 0\ 2018 = \{1..<2018*(2018 + 1) \text{ div } 2 + 1\}$

by *auto*

have *inj-on* (*uncurry f*) (*triangle 0 0 2018*)

proof (*rule eq-card-imp-inj-on*)

```

show finite (triangle 0 0 2018)
  by simp
next
  show card ((uncurry f) ‘ triangle 0 0 2018) = card (triangle 0 0 2018)
    using f(2) triangle-card
    by simp
qed

have path:  $\forall r0 < 2018. \forall c0 \leq r0. \forall n. r0 + n \leq 2018 \longrightarrow (\exists a b. a\ r0 =$ 
 $c0 \wedge b\ r0 = c0 \wedge$ 
 $(\forall r. r0 < r \wedge r < r0 + n \longrightarrow$ 
 $a\ r \neq b\ r \wedge c0 \leq a\ r \wedge a\ r \leq c0 + (r - r0) \wedge c0 \leq b\ r \wedge b$ 
 $r \leq c0 + (r - r0) \wedge$ 
 $(a\ r = b\ (r - 1) \vee a\ r = b\ (r - 1) + 1) \wedge$ 
 $(b\ r = b\ (r - 1) \vee b\ r = b\ (r - 1) + 1)) \wedge$ 
 $(\forall r. r0 \leq r \wedge r < r0 + n \longrightarrow f\ r\ (b\ r) = (\sum r' \leftarrow [r0..<r+1].$ 
 $f\ r'\ (a\ r'))))$  (is  $\forall r0 < 2018. \forall c0 \leq r0. \forall n. r0 + n \leq 2018 \longrightarrow ?P\ r0\ c0\ n$ )
proof safe
  fix r0 c0 n :: nat
  assume  $r0 < 2018\ c0 \leq r0\ r0 + n \leq 2018$ 
  then show  $?P\ r0\ c0\ n$ 
  proof (induction n)
    case 0
    then show ?case
      by auto
  next
    case (Suc n)

    show ?case
    proof (cases  $n = 0$ )
      case True
      then show ?thesis
        by auto
    next
      case False
      show ?thesis
      proof–
        obtain a b where *:
           $a\ r0 = c0\ b\ r0 = c0$ 
           $\forall r. r0 < r \wedge r < r0 + n \longrightarrow$ 

```

$$\begin{aligned}
& a\ r \neq b\ r \wedge \\
& c0 \leq a\ r \wedge a\ r \leq c0 + (r - r0) \wedge \\
& c0 \leq b\ r \wedge b\ r \leq c0 + (r - r0) \wedge \\
& (a\ r = b\ (r - 1) \vee a\ r = b\ (r - 1) + 1) \wedge \\
& (b\ r = b\ (r - 1) \vee b\ r = b\ (r - 1) + 1) \\
& \forall r. r0 \leq r \wedge r < r0 + n \longrightarrow f\ r\ (b\ r) = (\sum r' \leftarrow [r0..<r + 1]. f\ r'\ (a \\
& r')) \\
& \text{using } Suc \\
& \text{by } auto \\
& \text{have } ap': \forall r\ c. r0 \leq r \wedge r \leq r0 + n \wedge c0 \leq c \wedge c < c0 + (r - r0) \\
& \longrightarrow f\ (r - 1)\ c = |f\ r\ c - f\ r\ (c + 1)| \\
& \text{using } \langle antipascal\ f\ 2018 \rangle\ \langle n \neq 0 \rangle\ Suc(3-4) \\
& \text{unfolding } antipascal-def \\
& \text{by } auto \\
& \text{have } ap: f\ (r0 + n - 1)\ (b\ (r0 + n - 1)) = |f\ (r0 + n)\ (b\ (r0 + n - \\
& 1)) - f\ (r0 + n)\ (b\ (r0 + n - 1) + 1)| \\
& \text{proof } (cases\ n = 1) \\
& \text{case } True \\
& \text{then show } ?thesis \\
& \text{using } *(2)\ ap'[rule-format, of\ r0 + 1] \\
& \text{by } simp \\
& \text{next} \\
& \text{case } False \\
& \text{then have } n > 1 \\
& \text{using } \langle n \neq 0 \rangle \\
& \text{by } simp \\
& \text{show } ?thesis \\
& \text{proof } (subst\ ap') \\
& \text{have } r0 < r0 + n - 1 \\
& \text{using } \langle n > 1 \rangle \\
& \text{by } simp \\
& \text{then have } b\ (r0 + n - Suc\ 0) \leq c0 + n - Suc\ 0 \\
& \text{using } *(3)[rule-format, of\ r0 + n - 1]\ \langle n > 1 \rangle \\
& \text{by } simp \\
& \text{then show } r0 \leq r0 + n \wedge r0 + n \leq r0 + n \wedge c0 \leq b\ (r0 + n - 1) \\
& \wedge b\ (r0 + n - 1) < c0 + (r0 + n - r0) \\
& \text{using } *(3)[rule-format, of\ r0 + n - 1]\ \langle n > 1 \rangle \\
& \text{by } simp \\
& \text{qed } simp
\end{aligned}$$

qed

let $?an = \text{if } f(r0 + n) (b(r0 + n - 1)) < f(r0 + n) (b(r0 + n - 1) + 1) \text{ then } b(r0 + n - 1) \text{ else } b(r0 + n - 1) + 1$
let $?bn = \text{if } f(r0 + n) (b(r0 + n - 1)) < f(r0 + n) (b(r0 + n - 1) + 1) \text{ then } b(r0 + n - 1) + 1 \text{ else } b(r0 + n - 1)$
let $?a = a(r0 + n := ?an)$
let $?b = b(r0 + n := ?bn)$

have $?a\ r0 = c0\ ?b\ r0 = c0$
using $\langle n \neq 0 \rangle\ \langle a\ r0 = c0 \rangle\ \langle b\ r0 = c0 \rangle$
by *simp-all*

moreover

have $\forall r. r0 \leq r \wedge r < r0 + \text{Suc } n \longrightarrow f\ r\ (?b\ r) = (\sum\ r' \leftarrow [r0..<r+1]. f\ r' (?a\ r'))$

proof *safe*

fix r

assume $r0 \leq r \wedge r < r0 + \text{Suc } n$

show $f\ r\ (?b\ r) = (\sum\ r' \leftarrow [r0..<r+1]. f\ r' (?a\ r'))$

proof (*cases* $r < r0 + n$)

case *True*

then have $f\ r\ (?b\ r) = (\sum\ r' \leftarrow [r0..<r+1]. f\ r' (a\ r'))$

using $*(4)\ \langle r0 \leq r \rangle$

by *simp*

also have $\dots = (\sum\ r' \leftarrow [r0..<r+1]. f\ r' (?a\ r'))$

proof (*rule sum-list-cong, safe*)

fix r'

assume $r' \in \text{set } [r0..<r + 1]$

then show $f\ r' (a\ r') = f\ r' (?a\ r')$

using *True* $\langle r0 \leq r \rangle$

by *auto*

qed

finally show $?thesis$

by *simp*

next

case *False*

then have $r = r0 + n$

using $\langle r < r0 + \text{Suc } n \rangle$

```

    by simp
  show ?thesis
  proof (cases f (r0 + n) (b (r0 + n - 1)) < f (r0 + n) (b (r0 + n
- 1) + 1))
    case True
    have f (r0 + n) (b (r0 + n - 1) + 1) = f (r0 + n - 1) (b (r0 +
n - 1)) + f (r0 + n) (b (r0 + n - 1))
    using True ap
    by simp
    then have f (r0 + n) (b (r0 + n - 1) + 1) = (( $\sum r' \leftarrow [r0 .. < r0 + n]. f r' (a r')$ ) + f (r0 + n) (b (r0 + n - 1)))
    using *(4) (n ≠ 0)
    by simp
    also have ... = ( $\sum r' \leftarrow [r0 .. < r0 + n]. f r' (if r' = r0 + n then b (r0 + n - 1) else (a r'))$ ) + f (r0 + n) (if r0 + n = r0 + n then b (r0 + n - 1) else (a (r0 + n)))
    proof-
      have ( $\sum r' \leftarrow [r0 .. < r0 + n]. f r' (a r')$ ) = ( $\sum r' \leftarrow [r0 .. < r0 + n]. f r' (if r' = r0 + n then b (r0 + n - 1) else (a r'))$ )
      by (rule sum-list-cong, simp)
      then show ?thesis
      by simp
    qed
    also have ... = ( $\sum r' \leftarrow [r0 .. < r0 + n + 1]. f r' (if r' = r0 + n then b (r0 + n - 1) else (a r'))$ )
    by (subst sum-list-last, simp-all)
    finally show ?thesis
    using True (r = r0 + n)
    by simp (metis One-nat-def)
  next
    case False
    then have f (r0 + n) (b (r0 + n - 1)) = f (r0 + n - 1) (b (r0
+ n - 1)) + f (r0 + n) (b (r0 + n - 1) + 1)
    using ap
    by simp
    then have f (r0 + n) (b (r0 + n - 1)) = (( $\sum r' \leftarrow [r0 .. < r0 + n]. f r' (a r')$ ) + f (r0 + n) (b (r0 + n - 1) + 1))
    using *(4) (n ≠ 0)
    by simp
    also have ... = ( $\sum r' \leftarrow [r0 .. < r0 + n]. f r' (if r' = r0 + n then b (r0$ 

```

$+ n - 1) + 1 \text{ else } (a \ r')) + f \ (r0 + n) \ (if \ r0 + n = r0 + n \text{ then } b \ (r0 + n - 1) + 1 \text{ else } (a \ (r0 + n)))$
proof–
have $(\sum r' \leftarrow [r0..<r0 + n]. \ f \ r' \ (a \ r')) = (\sum r' \leftarrow [r0..<r0 + n]. \ f \ r' \ (if \ r' = r0 + n \text{ then } b \ (r0 + n - 1) + 1 \text{ else } (a \ r')))$
by $(rule \ sum-list-cong, \ simp)$
then show $?thesis$
by $simp$
qed
also have $\dots = (\sum r' \leftarrow [r0..<r0 + n + 1]. \ f \ r' \ (if \ r' = r0 + n \text{ then } b \ (r0 + n - 1) + 1 \text{ else } (a \ r')))$
by $(subst \ sum-list-last, \ simp-all)$
finally
show $?thesis$
using $False \ \langle r = r0 + n \rangle$
by $simp \ (metis \ One-nat-def \ Suc-eq-plus1)$
qed
qed
qed

moreover

have $\forall r. \ r0 < r \wedge r < r0 + Suc \ n \longrightarrow$
 $\quad ?a \ r \neq ?b \ r \wedge$
 $\quad c0 \leq ?a \ r \wedge ?a \ r \leq c0 + (r - r0) \wedge$
 $\quad c0 \leq ?b \ r \wedge ?b \ r \leq c0 + (r - r0) \wedge$
 $\quad (?a \ r = ?b \ (r - 1) \vee ?a \ r = ?b \ (r - 1) + 1) \wedge$
 $\quad (?b \ r = ?b \ (r - 1) \vee ?b \ r = ?b \ (r - 1) + 1)$

proof $safe$

fix r

assume $r0 < r \wedge r < r0 + Suc \ n \wedge ?a \ r = ?b \ r$

then show $False$

using $*$

by $(simp \ split: \ if-split-asm)$

next

fix r

assume $r0 < r \wedge r < r0 + Suc \ n$

show $c0 \leq ?a \ r$

proof $(cases \ r < r0 + n)$

case $True$

```

    then show ?thesis
      using * (r0 < r)
      by auto
  next
    case False
    then have r = r0 + n
      using (r < r0 + Suc n)
      by simp
    then show ?thesis
      using *(2) *(3)[rule-format, of r0 + n - 1]
      by (smt Suc-diff-1 Suc-eq-plus1 Suc-leD Suc-le-mono (r0 < r) add-gr-0
diff-less fun-upd-same less-antisym less-or-eq-imp-le zero-less-one)
    qed
  next
    fix r
    assume r0 < r r < r0 + Suc n
    show ?a r ≤ c0 + (r - r0)
    proof (cases r < r0 + n)
      case True
      then show ?thesis
        using * (r0 < r)
        by auto
    next
      case False
      then have r = r0 + n
        using (r < r0 + Suc n)
        by simp
      then show ?thesis
        using *(2) *(3)[rule-format, of r0 + n - 1]
        by (smt Suc-diff-Suc (r0 < r) add-Suc-right add-diff-cancel-left'
add-diff-cancel-right' fun-upd-same le-Suc-eq less-Suc-eq less-or-eq-imp-le nat-add-left-cancel-le
plus-1-eq-Suc)
    qed
  next
    fix r
    assume r0 < r r < r0 + Suc n
    show c0 ≤ ?b r
    proof (cases r < r0 + n)
      case True
      then show ?thesis

```

```

      using * ⟨r0 < r⟩
      by auto
    next
      case False
      then have r = r0 + n
        using ⟨r < r0 + Suc n⟩
        by simp
      then show ?thesis
        using *(2) *(3)[rule-format, of r0 + n - 1]
        by (smt Suc-diff-1 Suc-eq-plus1 Suc-leD Suc-le-mono ⟨r0 < r⟩ add-gr-0
diff-less fun-upd-same less-antisym less-or-eq-imp-le zero-less-one)
      qed
    next
      fix r
      assume r0 < r r < r0 + Suc n
      show ?b r ≤ c0 + (r - r0)
      proof (cases r < r0 + n)
        case True
        then show ?thesis
          using * ⟨r0 < r⟩
          by auto
      next
        case False
        then have r = r0 + n
          using ⟨r < r0 + Suc n⟩
          by simp
        then show ?thesis
          using *(2) *(3)[rule-format, of r0 + n - 1]
          by (smt Suc-diff-Suc ⟨r0 < r⟩ add-Suc-right add-diff-cancel-left'
add-diff-cancel-right' fun-upd-same le-Suc-eq less-Suc-eq less-or-eq-imp-le nat-add-left-cancel-le
plus-1-eq-Suc)
      qed
    next
      fix r
      assume r0 < r r < r0 + Suc n
        ?a r ≠ ?b (r - 1) + 1
      then show ?a r = ?b (r - 1)
        using *
        by (auto split: if-split-asm)
    next

```



```

    fix r
    assume r0 < r r < r0 + Suc n
      ?b r ≠ ?b (r - 1) + 1
    then show ?b r = ?b (r - 1)
      using *
      by (auto split: if-split-asm)
    qed
  ultimately
  show ?thesis
    by blast
  qed
qed
qed
qed

```

obtain $a\ b$ **where** $*$:

```

  a 0 = 0 b 0 = 0
  ∀ r. 0 < r ∧ r < 2018 ⟶ a r ≠ b r
  ∀ r. 0 < r ∧ r < 2018 ⟶ a r ≤ r
  ∀ r. 0 < r ∧ r < 2018 ⟶ b r ≤ r
  ∀ r. 0 < r ∧ r < 2018 ⟶ a r = b (r - 1) ∨ a r = b (r - 1) + 1
  ∀ r. 0 < r ∧ r < 2018 ⟶ b r = b (r - 1) ∨ b r = b (r - 1) + 1
  ∀ r < 2018. f r (b r) = (∑ r' ← [0..<r+1]. f r' (a r'))
  using path[rule-format, of 0 0 2018]
  by auto

```

have ab : $\forall r < 2018. a\ r = b\ r + 1 \vee a\ r = b\ r - 1$

using $*(1-3)\ *(6-7)$

by (*metis Suc-eq-plus1 diff-add-inverse diff-is-0-eq' le0 neq0-conv plus-1-eq-Suc*)

have $max-max$: $\forall r. 0 < r \wedge r < 2018 \longrightarrow max\ (a\ (r - 1))\ (b\ (r - 1)) \leq max\ (a\ r)\ (b\ r)$

proof *safe*

fix $r :: nat$

assume r : $0 < r\ r < 2018$

show $max\ (a\ (r - 1))\ (b\ (r - 1)) \leq max\ (a\ r)\ (b\ r)$

proof (*cases* $r = 1$)

case *True*

then show $?thesis$

using $\langle a\ 0 = 0 \rangle\ \langle b\ 0 = 0 \rangle$

```

    by simp
  next
    case False
    then have  $a\ r = b\ (r - 1) \vee a\ r = b\ (r - 1) + 1$ 
       $b\ r = b\ (r - 1) \vee b\ r = b\ (r - 1) + 1$ 
       $a\ r \neq b\ r \wedge a\ (r - 1) \neq b\ (r - 1)$ 
      using  $r *$ 
      by simp-all
    moreover
      have  $a\ (r - 1) = b\ (r - 1) \vee a\ (r - 1) = b\ (r - 1) + 1 \vee a\ (r - 1) =$ 
 $b\ (r - 1) - 1$ 
      using  $ab[rule-format, of\ r-1]\ r\ False$ 
      by auto
    ultimately
      show ?thesis
      by (smt diff-le-self eq-iff le-add1 max.commute max.mono)
  qed
qed

have min-min:  $\forall r. 0 < r \wedge r < 2018 \longrightarrow \min\ (a\ (r - 1))\ (b\ (r - 1)) \geq \min$ 
 $(a\ r)\ (b\ r) - 1$ 
  using  $*(2)\ *(3)\ *(6)\ *(7)$ 
  by (smt One-nat-def Suc-diff-Suc Suc-leD cancel-comm-monoid-add-class.diff-cancel
diff-zero le-0-eq le-diff-conv less-Suc-eq min.cobounded1 min-def nat-less-le)

let ?fa = map ( $\lambda r. f\ r\ (a\ r)$ )  $[0..<2018]$ 

have inj-on ( $\lambda r. f\ r\ (a\ r)$ ) (set  $[0..<2018]$ )
  unfolding inj-on-def
proof safe
  fix  $r1\ r2$ 
  assume  $r1 \in \text{set } [0..<2018]\ r2 \in \text{set } [0..<2018]$ 
   $f\ r1\ (a\ r1) = f\ r2\ (a\ r2)$ 
  have  $(r1, a\ r1) \in \text{triangle } 0\ 0\ 2018\ (r2, a\ r2) \in \text{triangle } 0\ 0\ 2018$ 
  using  $\langle r1 \in \text{set } [0..<2018]\ \rangle\ \langle r2 \in \text{set } [0..<2018]\ \rangle\ *(4)\ *(1)$ 
  using le-eq-less-or-eq triangle-def
  by auto
  moreover
    have  $f\ r1\ (a\ r1) = (\text{uncurry } f)\ (r1, a\ r1)\ f\ r2\ (a\ r2) = (\text{uncurry } f)\ (r2, a$ 
 $r2)$ 

```

```

    by auto
  ultimately
  show  $r1 = r2$ 
    using  $\langle inj-on (uncurry f) (triangle\ 0\ 0\ 2018) \rangle \langle f\ r1\ (a\ r1) = f\ r2\ (a\ r2) \rangle$ 
    by  $(metis\ inj-onD\ prod.inject)$ 
qed

```

```

have  $distinct\ ?fa$ 
  using  $\langle inj-on (\lambda r. f\ r\ (a\ r)) (set\ [0..<2018]) \rangle$ 
  by  $(simp\ add:\ distinct-map)$ 

```

```

have  $\forall x \in set\ ?fa. x > 0$ 
proof safe
  fix  $x$ 
  assume  $x \in set\ ?fa$ 
  then obtain  $r$  where  $r < 2018\ x = f\ r\ (a\ r)$ 
    by auto

```

```

have  $(r, a\ r) \in triangle\ 0\ 0\ 2018$ 
  using  $*(4)\ *(1)\ \langle r < 2018 \rangle$ 
  by  $(cases\ r = 0, auto\ simp\ add:\ triangle-def)$ 

```

moreover

```

have  $(uncurry f)\ (r, a\ r) = f\ r\ (a\ r)$ 
  by auto
ultimately
have  $f\ r\ (a\ r) \in (uncurry f)\ `triangle\ 0\ 0\ 2018$ 
  by  $(metis\ rev-image-eqI)$ 
then show  $x > 0$ 
  using  $f(2)\ \langle x = f\ r\ (a\ r) \rangle$ 
  by auto

```

qed

```

have  $set\ (map\ nat\ ?fa) = \{1..<2018+1\}$ 
proof  $(rule\ consecutive-nats)$ 
  show  $length\ (map\ nat\ ?fa) = 2018$ 
    by simp
next
  show  $distinct\ (map\ nat\ ?fa)$ 
  proof  $(subst\ distinct-map, safe)$ 
    show  $distinct\ (map\ (\lambda r. f\ r\ (a\ r))\ [0..<2018])$ 

```

```

      by fact
    next
      show inj-on nat (set ?fa)
        using ⟨∀ x ∈ set ?fa. x > 0⟩ inj-on-def
        by force
    qed
  next
    show ∀ x ∈ set (map nat ?fa). x > 0
      using ⟨∀ x ∈ set ?fa. x > 0⟩
      by simp
  next
    show sum-list (map nat ?fa) ≤ 2018 * (2018 + 1) div 2
  proof-
    have (∑ x ← ?fa. x) ∈ (uncurry f) ‘ (triangle 0 0 2018)
  proof-
    have (∑ x ← ?fa. x) = f 2017 (b 2017)
      using *(8)[rule-format, of 2017]
      by simp
    moreover
    have (2017, b 2017) ∈ triangle 0 0 2018
      using *(5)
      unfolding triangle-def
      by simp
    moreover
    have (uncurry f) (2017, b 2017) = f 2017 (b 2017)
      by simp
    ultimately
    show ?thesis
      by force
  qed
  then have (∑ x ← ?fa. x) ≤ 2018*(2018 + 1) div 2
    using f(2)
    by auto
  moreover
  have ∀ x ∈ {0..<2018}. 0 ≤ f x (a x)
    by (simp add: ⟨∀ x ∈ set (map (λr. f r (a r)) [0..<2018]). 0 < x⟩ le-less)
  ultimately
  show ?thesis
    using sum-list-nat[of [0..<2018] (λr. f r (a r))]
    by (simp add: comp-def)

```

```

    qed
  qed

  have ?fa <~> map int [1..<2018+1]
  proof-
    have set ?fa = set (map int [1..<2018+1])
    proof (subst inj-on-Un-image-eq-iff[symmetric])
      show nat ' set ?fa = nat ' set (map int [1..<2018+1])
      proof-
        have set (map nat ?fa) = nat ' set ?fa
        by auto
        moreover
        have nat ' set (map int [1..<2018+1]) = {1..<2018+1}
        by (metis (no-types, hide-lams) atLeastLessThan-upt map-idI map-map
nat-int o-apply set-map)
        ultimately
        show ?thesis
        using ⟨set (map nat ?fa) = {1..<2018+1}⟩
        by simp
      qed
    qed
  next
    show inj-on nat (set ?fa ∪ set (map int [1..<2018 + 1]))
    proof-
      have set ?fa ∪ set (map int [1..<2018 + 1]) ⊆ {x::int. x > 0}
      using ⟨∀ x ∈ set ?fa. x > 0⟩
      by auto
      moreover
      have inj-on nat {x::int. x > 0}
      by (metis inj-on-inverseI mem-Collect-eq nat-int zero-less-imp-eq-int)
      ultimately
      show ?thesis
      by (smt inj-onD inj-onI subsetD)
    qed
  qed

  then have mset ?fa = mset (map int [1..<2018+1])
  proof (subst set-eq-iff-mset-eq-distinct[symmetric])
    show distinct ?fa
    by fact
  next

```

```

    show distinct (map int [1..<2018+1])
    by (simp add: distinct-map)
qed simp

then show ?thesis
  using mset-eq-perm
  by blast
qed

let ?l = min (a 2017) (b 2017)
let ?r = max (a 2017) (b 2017)
let ?r0l = 2018 - ?l and ?c0l = 0 and ?nl = ?l
let ?r0r = ?r + 1 and ?c0r = ?r + 1 and ?nr = 2017 - ?r

{
  fix r0 c0 n
  assume triangle r0 c0 n  $\subseteq$  triangle 0 0 2018
  assume  $\forall r < 2018. (r, a\ r) \notin \text{triangle } r0\ c0\ n$ 
  assume  $n \geq 1008$ 
  assume  $c0 \leq r0\ r0 + n \leq 2018$ 

  have  $\forall p \in \text{triangle } r0\ c0\ n. (\text{uncurry } f)\ p > 2018$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then obtain r c where  $(r, c) \in \text{triangle } r0\ c0\ n$  and  $f\ r\ c \leq 2018$ 
    by auto
    moreover
    have  $(r, c) \in \text{triangle } 0\ 0\ 2018$ 
    using  $\langle \text{triangle } r0\ c0\ n \subseteq \text{triangle } 0\ 0\ 2018 \rangle \langle (r, c) \in \text{triangle } r0\ c0\ n \rangle$ 
    by auto
    then have  $f\ r\ c \geq 1$ 
    using  $\langle (\text{uncurry } f)\ ' (\text{triangle } 0\ 0\ 2018) = \{1..<2018*(2018 + 1) \text{ div } 2 + 1\} \rangle$ 
    by force
    then have  $\text{nat } (f\ r\ c) \in \{1..<2018+1\}$ 
    using  $\langle f\ r\ c \leq 2018 \rangle$ 
    by auto
    then have  $f\ r\ c \in \text{set } (\text{map } \text{int } [1..<2018+1])$ 
    by (smt  $\langle 1 \leq f\ r\ c \rangle \text{atLeastLessThan-upt image-eqI int-nat-eq set-map}$ )
    then have  $f\ r\ c \in \text{set } ?fa$ 

```

```

using ⟨?fa <~~> map int [1..<2018+1]>⟩
using perm-set-eq
by blast
then obtain  $r'$  where  $r' < 2018$   $f\ r' (a\ r') = f\ r\ c$ 
by auto
have  $(r', a\ r') \in \text{triangle } 0\ 0\ 2018$ 
using ⟨ $r' < 2018$ ⟩  $\ast(1)$   $\ast(4)$ 
by (cases  $r' = 0$ ) (auto simp add: triangle-def)
then have  $r = r'\ c = a\ r'$ 
using ⟨ $f\ r' (a\ r') = f\ r\ c$ ⟩ ⟨inj-on (uncurry f) (triangle 0 0 2018)⟩ ⟨ $(r, c) \in$ 
triangle 0 0 2018⟩
unfolding inj-on-def
by force+

then have  $(r', a\ r') \in \text{triangle } r0\ c0\ n$ 
using ⟨ $(r, c) \in \text{triangle } r0\ c0\ n$ ⟩
by simp

then show False
using ⟨ $r' < 2018$ ⟩ ⟨ $\forall\ r < 2018. (r, a\ r) \notin \text{triangle } r0\ c0\ n$ ⟩
by auto
qed

obtain  $ar\ br$  where  $r$ :
 $ar\ r0 = c0\ br\ r0 = c0$ 
 $\forall r. r0 < r \wedge r < r0 + n \longrightarrow ar\ r \neq br\ r \wedge$ 
 $c0 \leq ar\ r \wedge ar\ r \leq c0 + (r - r0) \wedge$ 
 $c0 \leq br\ r \wedge br\ r \leq c0 + (r - r0) \wedge$ 
 $(ar\ r = br\ (r - 1) \vee ar\ r = (br\ (r - 1)) + 1) \wedge$ 
 $(br\ r = br\ (r - 1) \vee br\ r = (br\ (r - 1)) + 1)$ 
 $\forall r. r0 \leq r \wedge r < r0 + n \longrightarrow$ 
 $f\ r\ (br\ r) =$ 
 $(\sum_{r' \leftarrow [r0..<r+1]}. f\ r' (ar\ r'))$ 
using ⟨ $r0 + n \leq 2018$ ⟩ ⟨ $c0 \leq r0$ ⟩ ⟨ $n \geq 1008$ ⟩
using path[rule-format, of  $r0\ c0\ n$ ]
by auto

have  $\forall\ r. r0 \leq r \wedge r < r0 + n \longrightarrow (r, ar\ r) \in \text{triangle } r0\ c0\ n$ 
proof safe
fix  $r$ 

```

```

assume  $r0 \leq r \wedge r < r0 + n$ 
then show  $(r, ar\ r) \in triangle\ r0\ c0\ n$ 
  using  $r(1)\ r(2)\ r(3)[rule-format, of\ r]$ 
  unfolding triangle-def
  by (cases  $r = r0$ ) auto
qed
then have  $\forall\ r. r0 \leq r \wedge r < r0 + n \longrightarrow f\ r\ (ar\ r) > 2018$ 
  using  $\langle \forall\ p \in triangle\ r0\ c0\ n. (uncurry\ f)\ p > 2018 \rangle$ 
  by force

have  $(r0 + n - 1, br\ (r0 + n - 1)) \in triangle\ r0\ c0\ n$ 
  using  $r(3)[rule-format, of\ r0 + n - 1]$ 
  using  $\langle r0 + n \leq 2018 \rangle\ \langle n \geq 1008 \rangle$ 
  by (simp add: triangle-def)
then have  $(r0 + n - 1, br\ (r0 + n - 1)) \in triangle\ 0\ 0\ 2018$ 
  using  $\langle triangle\ r0\ c0\ n \subseteq triangle\ 0\ 0\ 2018 \rangle$ 
  by blast
then have  $(uncurry\ f)\ (r0 + n - 1, br\ (r0 + n - 1)) \in \{1..<2018 * (2018$ 
 $+ 1) \div 2 + 1\}$ 
  using  $f(2)$ 
  by blast
then have  $f\ (r0 + n - 1)\ (br\ (r0 + n - 1)) \leq 2018 * (2018 + 1) \div 2$ 
  by simp

moreover

  have  $f\ (r0 + n - 1)\ (br\ (r0 + n - 1)) = (\sum r' \leftarrow [r0..<(r0 + n - 1) + 1].$ 
 $f\ r'\ (ar\ r'))$ 
  using  $r(4)[rule-format, of\ r0 + n - 1]$ 
  using  $\langle r0 + n \leq 2018 \rangle\ \langle n \geq 1008 \rangle$ 
  by simp

ultimately

have  $1: (\sum r' \leftarrow [r0..<(r0 + n - 1) + 1]. f\ r'\ (ar\ r')) \leq 2018 * (2018 + 1) \div$ 
 $2$ 
  by simp

have  $length\ ([r0..<(r0 + n - 1) + 1]) = n$ 
  using  $\langle n \geq 1008 \rangle$ 

```



```

by auto

have n * (2 * 2018 + n + 1) div 2 ≥ 1008 * (2*2018 + 1008 + 1) div 2
proof-
  have n * (2 * 2018 + n + 1) ≥ 1008 * (2*2018 + 1008 + 1)
    using ⟨n ≥ 1008⟩
    by (metis Suc-eq-plus1 add-Suc mult-le-mono nat-add-left-cancel-le)
  then show ?thesis
    using div-le-mono
    by blast
qed

moreover

have length [r0..<(r0 + n - 1) + 1] * (2 * 2018 + length [r0..<(r0 + n -
1) + 1] + 1) div 2 ≤
  (∑ r'←[r0..<(r0 + n - 1) + 1]. nat (f r' (ar r')))
proof (rule sum-list-distinct-lb)
  have ∀ r'∈set [r0..<(r0 + n - 1) + 1]. 2018 < f r' (ar r')
    using ⟨∀ r. r0 ≤ r ∧ r < r0 + n ⟶ f r (ar r) > 2018⟩ ⟨n ≥ 1008⟩
    by simp
  then show ∀ r'∈set [r0..<(r0 + n - 1) + 1]. 2018 < nat (f r' (ar r'))
    by auto
next
show distinct (map (λx. nat (f x (ar x))) [r0..<(r0 + n - 1) + 1])
proof (subst distinct-map, safe)
  show inj-on (λx. nat (f x (ar x))) (set [r0..<(r0 + n - 1) + 1])
    unfolding inj-on-def
  proof safe
    fix r1 r2
    assume r1 ∈ set [r0..<(r0 + n - 1) + 1] r2 ∈ set [r0..<(r0 + n - 1)
+ 1]
      nat (f r1 (ar r1)) = nat (f r2 (ar r2))
    have (r1, ar r1) ∈ triangle r0 c0 n (r2, ar r2) ∈ triangle r0 c0 n
      using ⟨r1 ∈ set [r0..<(r0 + n - 1) + 1]⟩ ⟨r2 ∈ set [r0..<(r0 + n -
1) + 1]⟩
    using ⟨∀ r. r0 ≤ r ∧ r < r0 + n ⟶ (r, ar r) ∈ triangle r0 c0 n⟩
    using ⟨n ≥ 1008⟩
    by force+

```

```

then have  $(r1, ar\ r1) \in triangle\ 0\ 0\ 2018$   $(r2, ar\ r2) \in triangle\ 0\ 0\ 2018$ 
using  $\langle triangle\ r0\ c0\ n \subseteq triangle\ 0\ 0\ 2018 \rangle$ 
by blast+

moreover

have  $f\ r1\ (a\ r1) = (uncurry\ f)\ (r1, a\ r1)$   $f\ r2\ (a\ r2) = (uncurry\ f)\ (r2,$ 
 $a\ r2)$ 
by auto

moreover

have  $f\ r1\ (a\ r1) = f\ r2\ (a\ r2)$ 
using  $\langle (r1, ar\ r1) \in triangle\ 0\ 0\ 2018 \rangle \langle (r2, ar\ r2) \in triangle\ 0\ 0\ 2018 \rangle$ 
using  $\langle nat\ (f\ r1\ (ar\ r1)) = nat\ (f\ r2\ (ar\ r2)) \rangle$ 
using  $\langle (r1, ar\ r1) \in triangle\ r0\ c0\ n \rangle$ 
using  $\langle \forall\ p \in triangle\ r0\ c0\ n. 2018 < uncurry\ f\ p \rangle$ 
using  $\langle inj\text{-}on\ (uncurry\ f)\ (triangle\ 0\ 0\ 2018) \rangle$ 
by  $(smt\ Pair\text{-}inject\ eq\text{-}nat\text{-}nat\text{-}iff\ inj\text{-}on\text{-}def\ nat\text{-}0\text{-}iff\ uncurry.simps)$ 

ultimately

show  $r1 = r2$ 
using  $\langle inj\text{-}on\ (uncurry\ f)\ (triangle\ 0\ 0\ 2018) \rangle$ 
using  $\langle (r1, ar\ r1) \in triangle\ r0\ c0\ n \rangle$ 
 $\langle \forall\ p \in triangle\ r0\ c0\ n. 2018 < uncurry\ f\ p \rangle$ 
 $\langle nat\ (f\ r1\ (ar\ r1)) = nat\ (f\ r2\ (ar\ r2)) \rangle$ 
by  $(smt\ Pair\text{-}inject\ inj\text{-}on\text{-}eq\text{-}iff\ int\text{-}nat\text{-}eq\ uncurry.simps)$ 
qed
qed simp
qed

ultimately

have  $(\sum\ r' \leftarrow [r0..<(r0 + n - 1) + 1].\ nat\ (f\ r'\ (ar\ r')))) \geq 1008 * (2*2018$ 
 $+ 1008 + 1)\ div\ 2$ 
using  $\langle length\ ([r0..<(r0 + n - 1) + 1]) = n \rangle$ 
by simp

moreover

```

```

have ( $\sum r' \leftarrow [r0..<(r0 + n - 1) + 1]. \text{nat } (f r' (ar r'))$ ) =  $\text{nat } ((\sum r' \leftarrow [r0..<(r0$ 
+  $n - 1) + 1]. f r' (ar r'))$ )
proof (rule sum-list-nat)
  show  $\forall r' \in \text{set } [r0..<(r0 + n - 1) + 1]. 0 \leq f r' (ar r')$ 
    using  $\langle \forall r. r0 \leq r \wedge r < r0 + n \longrightarrow f r (ar r) > 2018 \rangle \langle n \geq 1008 \rangle$ 
    by auto
qed

```

ultimately

```

have 2:  $\text{nat } ((\sum r' \leftarrow [r0..<(r0 + n - 1) + 1]. f r' (ar r')) \geq 1008 * (2 * 2018$ 
+  $1008 + 1) \text{ div } 2$ 
by simp

```

```

have False
  using 1 2
  by simp
} note triangle = this

```

```

show False
proof (cases ?nl ≤ ?nr)
  case True
    show False
    proof (rule triangle)
      show triangle ?r0r ?c0r ?nr ⊆ triangle 0 0 2018
        unfolding triangle-def
        by auto
    next
      show ?nr ≥ 1008
        using ab[rule-format, of 2017] True
        by (auto simp add: max-def min-def split: if-split-asm)
    next
      show  $\forall r < 2018. (r, a r) \notin \text{triangle } ?r0r ?c0r ?nr$ 
      proof–
        have  $\forall r < 2018. \max (a r) (b r) \leq ?r$ 
        proof–
          have  $\forall r < 2018. \max (a (2017 - r)) (b (2017 - r)) \leq ?r$ 
          proof safe

```

```

fix r::nat
assume r < 2018
then show max (a (2017 - r)) (b (2017 - r)) ≤ ?r
proof (induction r)
  case 0
  then show ?case
  by simp
next
  case (Suc r)
  then show ?case
  using max-max *(1-2)
  by (smt Suc-diff-Suc Suc-lessD add-diff-cancel-left' diff-Suc-Suc
diff-less-Suc max.boundedE max.orderE one-plus-numeral plus-1-eq-Suc semiring-norm(4)
semiring-norm(5) zero-less-diff)
  qed
  qed
  then show ?thesis
  by (metis Suc-leI add-le-cancel-left diff-diff-cancel diff-less-Suc one-plus-numeral
plus-1-eq-Suc semiring-norm(4) semiring-norm(5))
  qed
  then show ?thesis
  unfolding triangle-def
  by auto
  qed
next
  show ?r0r ≤ ?c0r
  by simp
next
  show ?r0r + ?nr ≤ 2018
  by (simp add: *(4) *(5))
  qed
next
  case False
  show ?thesis
  proof (rule triangle)
    show triangle ?r0l ?c0l ?nl ⊆ triangle 0 0 2018
    using *(4)[rule-format, of 2017] *(5)[rule-format, of 2017]
    unfolding triangle-def
    by auto
  next

```

```

show ?c0l ≤ ?r0l
  by simp
next
show 2018 − min (a 2017) (b 2017) + min (a 2017) (b 2017) ≤ 2018
  using *(4)[rule-format, of 2017] *(5)[rule-format, of 2017]
  by auto
next
show ?nl ≥ 1008
  using ab[rule-format, of 2017] False
  by (auto simp add: max-def min-def split: if-split-asm)
next
show ∀ r < 2018. (r, a r) ∉ triangle ?r0l ?c0l ?nl
proof−
  have ∀ r < 2018. min (a r) (b r) ≥ ?l − (2017 − r)
proof−
  have ∀ r < 2018. min (a (2017 − r)) (b (2017 − r)) ≥ ?l − (2017 −
(2017 − r))
  proof safe
    fix r::nat
    assume r < 2018
    then show ?l − (2017 − (2017 − r)) ≤ min (a (2017 − r)) (b (2017
− r))
  proof (induction r)
    case 0
    then show ?case
    by simp
  next
    case (Suc r)
    have min (a 2017) (b 2017) − (2017 − (2017 − Suc r)) = min (a
2017) (b 2017) − r − 1
    using ⟨Suc r < 2018⟩
    by auto
    also have ... ≤ min (a (2017 − r)) (b (2017 − r)) − 1
    using Suc
    by (smt Suc-lessD diff-Suc-Suc diff-diff-cancel diff-le-mono le-less
one-plus-numeral plus-1-eq-Suc semiring-norm(4) semiring-norm(5) zero-less-diff)
    also have ... ≤ min (a (2017 − r − 1)) (b (2017 − r − 1))
    using min-min[rule-format, of 2017 − r] ⟨Suc r < 2018⟩
    by simp
  finally

```

```

      show ?case
      by simp
    qed
  qed
  then show ?thesis
    by (smt diff-diff-cancel diff-less-Suc le-less less-Suc-eq one-plus-numeral
plus-1-eq-Suc semiring-norm(4) semiring-norm(5))
  qed
  then show ?thesis
    by (auto simp add: triangle-def)
  qed
  qed
  qed
  qed
end

```

5.3 Number theory problems

5.3.1 IMO 2018 SL - N5

```

theory IMO-2018-SL-N5-sol
imports Main
begin

```

```

definition perfect-square :: int  $\Rightarrow$  bool where
  perfect-square s  $\longleftrightarrow$  ( $\exists$  r. s = r * r)

```

```

lemma perfect-square-root-pos:
  assumes perfect-square s
  shows  $\exists$  r. r  $\geq$  0  $\wedge$  s = r * r
  using assms
  unfolding perfect-square-def
  by (smt mult-minus-left mult-minus-right)

```

```

lemma not-perfect-square-15:
  fixes q::int
  shows q2  $\neq$  15
proof (rule ccontr)
  assume  $\neg$  ?thesis

```

```

then have  $3^2 < (\text{abs } q)^2 (\text{abs } q)^2 < 4^2$ 
  by auto
then have  $3 < \text{abs } q \text{ abs } q < 4$ 
  using abs-ge-zero power-less-imp-less-base zero-le-numeral
  by blast+
then show False
  by simp
qed

```

```

lemma not-perfect-square-12:
  fixes  $q::\text{int}$ 
  shows  $q^2 \neq 12$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $3^2 < (\text{abs } q)^2 (\text{abs } q)^2 < 4^2$ 
    by auto
  then have  $3 < \text{abs } q \text{ abs } q < 4$ 
    using abs-ge-zero power-less-imp-less-base zero-le-numeral
    by blast+
  then show False
    by simp
qed

```

```

lemma not-perfect-square-8:
  fixes  $q::\text{int}$ 
  shows  $q^2 \neq 8$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $2^2 < (\text{abs } q)^2 (\text{abs } q)^2 < 3^2$ 
    by auto
  then have  $2 < \text{abs } q \text{ abs } q < 3$ 
    using abs-ge-zero power-less-imp-less-base zero-le-numeral
    by blast+
  then show False
    by simp
qed

```

```

lemma not-perfect-square-7:
  fixes  $q::\text{int}$ 
  shows  $q^2 \neq 7$ 

```

```

proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $2^2 < (abs\ q)^2 \wedge (abs\ q)^2 < 3^2$ 
    by auto
  then have  $2 < abs\ q \wedge abs\ q < 3$ 
    using abs-ge-zero power-less-imp-less-base zero-le-numeral
    by blast+
  then show False
    by simp
qed

```

```

lemma not-perfect-square-5:
  fixes  $q::int$ 
  shows  $q^2 \neq 5$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $2^2 < (abs\ q)^2 \wedge (abs\ q)^2 < 3^2$ 
    by auto
  then have  $2 < abs\ q \wedge abs\ q < 3$ 
    using abs-ge-zero power-less-imp-less-base zero-le-numeral
    by blast+
  then show False
    by simp
qed

```

```

lemma not-perfect-square-3:
  fixes  $q::int$ 
  shows  $q^2 \neq 3$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $1^2 < (abs\ q)^2 \wedge (abs\ q)^2 < 2^2$ 
    by auto
  then have  $1 < abs\ q \wedge abs\ q < 2$ 
    using abs-ge-zero power-less-imp-less-base zero-le-numeral
    by blast+
  then show False
    by simp
qed

```

```

lemma IMO2018SL-N5-lemma:

```



```

fixes  $s\ a\ b\ c\ d :: \text{int}$ 
assumes  $s^2 = a^2 + b^2$   $s^2 = c^2 + d^2$   $2*s = a^2 - c^2$ 
assumes  $s > 0$   $a \geq 0$   $d \geq 0$   $b \geq 0$   $c \geq 0$   $b > 0 \vee c > 0$   $b \geq c$ 
shows False
proof–
  have  $2*s = d^2 - b^2$ 
    using assms
    by simp

  have  $d > 0$ 
    using  $\langle 2 * s = d^2 - b^2 \rangle \langle s > 0 \rangle \langle d \geq 0 \rangle$ 
    by (smt pos-imp-zdiv-neg-iff zero-less-power2)

  have  $a > 0$ 
    using  $\langle 2 * s = a^2 - c^2 \rangle \langle s > 0 \rangle \langle a \geq 0 \rangle$ 
    by (smt pos-imp-zdiv-neg-iff zero-less-power2)

  have  $b > 0$ 
    using assms
    by auto

  have  $d^2 > c^2$ 
    using  $\langle 2 * s = d^2 - b^2 \rangle \langle c \leq b \rangle \langle 0 < s \rangle \langle c \geq 0 \rangle$ 
    by (smt power-mono)

  then have  $d^2 > s^2 \text{ div } 2$ 
    using  $\langle s^2 = c^2 + d^2 \rangle$ 
    by presburger

  then have  $2*s^2 < 4*d^2$ 
    by simp

  have  $b < d$ 
    using  $\langle 2*s = d^2 - b^2 \rangle \langle s > 0 \rangle \langle d > 0 \rangle \langle b > 0 \rangle$ 
    by (smt power-mono-iff zero-less-numeral)

  have even  $b \longleftrightarrow$  even  $d$ 
    using  $\langle 2*s = d^2 - b^2 \rangle$ 
    by (metis add-uminus-conv-diff dvd-minus-iff even-add even-mult-iff even-numeral
power2-eq-square)

```

```

then have  $b \leq d - 2$ 
  using  $\langle b < d \rangle$ 
  by (smt even-two-times-div-two odd-two-times-div-two-succ)

then have  $2*s \geq d^2 - (d-2)^2$ 
  using  $\langle 2*s = d^2 - b^2 \rangle \langle d > 0 \rangle \langle b > 0 \rangle$ 
  by auto
then have  $s \geq 2*(d - 1)$ 
  by (simp add: algebra-simps power2-eq-square)
then have  $2*d \leq s + 2$ 
  by simp
then have  $4*d^2 \leq (s + 2)^2$ 
  using abs-le-square-iff[of  $2*d$   $s + 2$ ]  $\langle d > 0 \rangle \langle s > 0 \rangle$ 
  by auto
then have  $2*s^2 < (s+2)^2$ 
  using  $\langle 2*s^2 < 4*d^2 \rangle$ 
  by simp
then have  $(s - 2)^2 < 8$ 
  by (simp add: power2-eq-square algebra-simps)
then have  $(s - 2)^2 < 3^2$ 
  by simp
then have  $s - 2 < 3$ 
  using power2-less-imp-less
  by fastforce
then have  $s \leq 4$ 
  by simp
then have  $s = 1 \vee s = 2 \vee s = 3 \vee s = 4$ 
  using  $\langle s > 0 \rangle$ 
  by auto
moreover
have  $\bigwedge p\ q :: \text{int}. \llbracket 16 = p^2 + q^2; p \geq 0; q \geq 0 \rrbracket \implies p = 0 \vee q = 0$ 
proof-
  fix  $p\ q :: \text{int}$ 
  assume  $16 = p^2 + q^2\ p \geq 0\ q \geq 0$ 
  have  $p \leq 4$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then have  $p \geq 5$ 
    by simp

```

```

then have  $p^2 \geq 25$ 
  using power-mono[of 5 p 2]
  by simp
then have  $p^2 + q^2 \geq 25$ 
  using zero-le-power2[of q]
  by linarith
then show False
  using  $\langle 16 = p^2 + q^2 \rangle$ 
  by auto
qed
then have  $p = 0 \vee p = 1 \vee p = 2 \vee p = 3 \vee p = 4$ 
  using  $\langle 0 \leq p \rangle$ 
  by auto
then show  $p = 0 \vee q = 0$ 
  using  $\langle 16 = p^2 + q^2 \rangle$  not-perfect-square-15 not-perfect-square-12 not-perfect-square-7
  by auto
qed
moreover
have  $\bigwedge p\ q :: \text{int. } \llbracket 9 = p^2 + q^2; p \geq 0; q \geq 0 \rrbracket \implies p = 0 \vee q = 0$ 
proof–
  fix  $p\ q :: \text{int}$ 
  assume  $9 = p^2 + q^2\ p \geq 0\ q \geq 0$ 
  have  $p \leq 3$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then have  $p \geq 4$ 
    by simp
    then have  $p^2 \geq 16$ 
    using power-mono[of 4 p 2]
    by simp
    then have  $p^2 + q^2 \geq 16$ 
    using zero-le-power2[of q]
    by linarith
    then show False
    using  $\langle 9 = p^2 + q^2 \rangle$ 
    by auto
  qed
then have  $p = 0 \vee p = 1 \vee p = 2 \vee p = 3$ 
  using  $\langle 0 \leq p \rangle$ 
  by auto

```

```

    then show  $p = 0 \vee q = 0$ 
      using  $\langle 9 = p^2 + q^2 \rangle$  not-perfect-square-8 not-perfect-square-5
      by auto
qed
moreover
have  $\bigwedge p\ q :: \text{int. } \llbracket 4 = p^2 + q^2; p \geq 0; q \geq 0 \rrbracket \implies p = 0 \vee q = 0$ 
proof-
  fix  $p\ q :: \text{int}$ 
  assume  $4 = p^2 + q^2\ p \geq 0\ q \geq 0$ 
  have  $p \leq 2$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then have  $p \geq 3$ 
      by simp
    then have  $p^2 \geq 9$ 
      using power-mono[of 3 p 2]
      by simp
    then have  $p^2 + q^2 \geq 9$ 
      using zero-le-power2[of q]
      by linarith
    then show False
      using  $\langle 4 = p^2 + q^2 \rangle$ 
      by auto
  qed
  then have  $p = 0 \vee p = 1 \vee p = 2$ 
    using  $\langle 0 \leq p \rangle$ 
    by auto
  then show  $p = 0 \vee q = 0$ 
    using  $\langle 4 = p^2 + q^2 \rangle$  not-perfect-square-3
    by auto
qed
moreover
have  $\bigwedge p\ q :: \text{int. } \llbracket 1 = p^2 + q^2; p \geq 0; q \geq 0 \rrbracket \implies p = 0 \vee q = 0$ 
  by (smt one-le-power)
moreover
have  $a \neq 0\ d \neq 0$ 
  using  $\langle a > 0 \rangle\ \langle d > 0 \rangle$ 
  by auto
ultimately
have  $c = 0\ b = 0$ 

```

```

    using ⟨s2 = c2 + d2⟩ ⟨d ≥ 0⟩ ⟨c ≥ 0⟩ ⟨s2 = a2 + b2⟩ ⟨a ≥ 0⟩ ⟨b ≥ 0⟩
    by fastforce+
  then show False
    using ⟨b > 0 ∨ c > 0⟩
    by auto
qed

```

theorem *IMO2018SL-N5*:

```

  fixes x y z t :: int
  assumes pos: x > 0 y > 0 z > 0 t > 0
  assumes eq: x*y - z*t = x + y x + y = z + t
  shows ¬ (perfect-square (x*y) ∧ perfect-square (z*t))
proof (rule ccontr)
  assume ¬ ?thesis
  then obtain a c where x*y = a*a z*t = c*c a > 0 c > 0
    using perfect-square-root-pos pos
    by (smt zero-less-mult-iff)

```

show False

```

proof (cases odd (x + y))
case True

```

```

  have even (x * y)
    using True
    by auto

```

moreover

```

  have odd (z + t)
    using True eq(2)
    by simp
  then have even (z * t)
    by auto

```

ultimately

```

  have even (x*y - z*t)
    by simp
  then show False
    using eq(1) True

```

```

    by simp
next
case False
then have even (x + y) even (z + t)
    using eq(2)
    by auto

let ?s = (x + y) div 2
let ?b = abs (x - y) div 2 and ?d = abs (z - t) div 2

have ?s ^ 2 = a ^ 2 + ?b ^ 2
proof-
  have a^2 + ?b^2 = (x+y)^2 div 4
    using ⟨even (x+y)⟩ div-power[of 2 abs (x - y) 2] ⟨x*y = a*a⟩
    by (simp add: power2-eq-square algebra-simps)
  then show ?thesis
    by (metis False div-power mult-2-right numeral-Bit0 power2-eq-square)
qed

have ?s ^ 2 = c ^ 2 + ?d ^ 2
proof-
  have c^2 + ?d^2 = (z+t)^2 div 4
    using ⟨even (z+t)⟩ div-power[of 2 abs (z - t) 2] ⟨z*t = c*c⟩
    by (simp add: power2-eq-square algebra-simps)
  then show ?thesis
    by (metis eq(2) False div-power mult-2-right numeral-Bit0 power2-eq-square)
qed

have 2*?s = a^2 - c^2
  using ⟨even (x + y)⟩ ⟨x*y = a*a⟩ ⟨z*t = c*c⟩ eq(1)
  by (simp add: power2-eq-square)

have ?s > 0
  using ⟨x > 0⟩ ⟨y > 0⟩
  by auto

have ?b ≥ 0 ?d ≥ 0
  by simp-all

show ?thesis

```

```

proof (cases ?b ≥ c)
  case True
  then show False
    using IMO2018SL-N5-lemma[of ?s a ?b c ?d]
    using ⟨?s2 = a2 + ?b2⟩ ⟨?s2 = c2 + ?d2⟩ ⟨2*?s = a2 - c2⟩
    using ⟨a > 0⟩ ⟨c > 0⟩ ⟨?s > 0⟩ ⟨?d ≥ 0⟩
    by simp
  next
  case False
  then have c ≥ ?b
    by simp
  then show False
    using IMO2018SL-N5-lemma[of ?s ?d c ?b a]
    using ⟨?s2 = a2 + ?b2⟩ ⟨?s2 = c2 + ?d2⟩ ⟨2*?s = a2 - c2⟩
    using ⟨a > 0⟩ ⟨c > 0⟩ ⟨?s > 0⟩ ⟨?b ≥ 0⟩ ⟨?d ≥ 0⟩
    by simp
  qed
qed
qed
end

```