

```

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

```

A mathematical frog jumps along the number line. The frog starts at 1, and jumps according to the following rule: if the frog is at integer  $n$ , then it can jump to  $n + 1$  or  $n + 2^{m_n+1}$ , where  $2^{m_n}$  is the largest power of 2 that is a factor of  $n$ . Show that if  $k \geq 2$ , is a positive integer and  $i$  is a non-negative integer, then the minimum number of jumps needed to reach  $2^i k$  is greater than the minimum number of jumps needed to reach  $2^i$ .

```

definition jump :: nat  $\Rightarrow$  nat where
    jump n = n + 2(multiplicity 2 n + 1)

```

*reachable s n* is true if the frog can reach integer  $n$  with  $s$  jumps.

```

inductive reachable :: nat  $\Rightarrow$  nat  $\Rightarrow$  bool where
    Start[intro]: reachable 0 1 |
    Incr[intro]: reachable s n  $\implies$  reachable (Suc s) (Suc n) |
    Jump[intro]: reachable s n  $\implies$  reachable (Suc s) (jump n)

```

The *height* of a number is the minimum number of jumps required.

```

definition height :: nat  $\Rightarrow$  nat where
    height n = arg-min id ( $\lambda s$ . reachable s n)

```

## 0.1 Alternative definition of *jump*

*jump 0* isn't really well-defined. We introduce an alternative definition:

```

definition jump' where
    jump' n = (if n = 0 then 0 else jump n)

```

```

lemma jump': n  $\neq$  0  $\implies$  jump n = jump' n
unfolding jump-def jump'-def by simp

```

```

lemma jump'-0[simp]: jump' 0 = 0 by (simp add: jump'-def)

```

```

inductive reachable' :: nat  $\Rightarrow$  nat  $\Rightarrow$  bool where
    Start[intro]: reachable' 0 1 |
    Incr[intro]: reachable' s n  $\implies$  reachable' (Suc s) (Suc n) |
    Jump[intro]: reachable' s n  $\implies$  reachable' (Suc s) (jump' n)

```

```

lemma zero-unreachable:

```

```

     $\neg$ reachable s 0

```

```

proof

```

```

    assume reachable s 0

```

```

    moreover have jump n > 0 for n unfolding jump-def by simp

```

```

    ultimately show False

```

```

    apply (cases rule: reachable.cases)

```

```

    apply presburger

```

```

    by (metis not-less0)

```

```

qed

```

```

lemma reachable':

```

```

    reachable s n = reachable' s n

```

```

proof

```

```

    assume reachable s n

```

```

    then show reachable' s n

```

```

    proof (induction rule: reachable.induct)

```

```

        case (Jump s n)

```

```

        from (reachable s n) have n  $\neq$  0

```

```

        by (metis zero-unreachable)

```

```

        with Jump show ?case by (auto simp add: jump')

```

```

qed (intro reachable'.intros)+
next
assume reachable' s n
then show reachable s n
proof (induction rule: reachable'.induct)
case (Jump s n)
from (reachable s n) have n ≠ 0
by (metis zero-unreachable)
with Jump show ?case by (auto simp flip: jump')
qed (intro reachable.intros)+
qed

```

```

lemma zero-unreachable': ¬ reachable' s 0
using zero-unreachable by (simp add: reachable')

```

```

definition height' :: nat ⇒ nat where
height' n = arg-min id (λs. reachable' s n)

```

```

lemma height'[simp]:
height n = height' n
unfolding height-def height'-def by (simp add: reachable')

```

## 0.2 Properties of $jump'$

```

lemma jump-even[simp]: jump' (2*n) = 2 * jump' n
by (auto simp add: jump'-def jump-def multiplicity-times-same)

```

```

lemma jump-odd[simp]: jump' (Suc (2*n)) = Suc (Suc (Suc (2*n)))
unfolding jump'-def jump-def
by (auto intro: not-dvd-imp-multiplicity-0)

```

```

lemma jump-gt0[simp]: n > 0 ⇒ jump' n > 0
unfolding jump'-def jump-def by auto

```

## 0.3 Representing paths

```

lemma reachable-height:
n ≠ 0 ⇒ reachable' (height' n) n
— This is not a tautology, we need to prove that the number is reachable at all.
unfolding height'-def
proof (intro arg-min-natI)
assume n ≠ 0
then show reachable' (n - 1) n
proof (induction n)
case (Suc n)
then show ?case
using Start Incr by (cases n) auto
qed auto
qed

```

Based on this, we define a notion of a path a specific set of moves with a desired result.

```

datatype move = Incr | Jump
type-synonym path = move list

```

```

fun make-move :: move ⇒ nat ⇒ nat where
make-move Incr n = Suc n |
make-move Jump n = jump' n

```

```

fun make-moves :: path ⇒ nat ⇒ nat where
make-moves moves n = fold make-move moves n

```

```

lemma path-exists:
assumes reachable' s n

```

```

  shows  $\exists \text{ path. make-moves path } 1 = n \wedge \text{length path} = s$ 
  using assms
proof induction
  case (Incr s n)
  then obtain path where make-moves path 1 = n and length path = s
    by auto
  hence make-moves (path @ [Incr]) 1 = Suc n and length (path @ [Incr]) = Suc s
    by auto
  then show ?case by blast
next
  case (Jump s n)
  then obtain path where make-moves path 1 = n and length path = s
    by auto
  moreover have  $n \neq 0$  using Jump.hyps zero-unreachable' by meson
  ultimately have make-moves (path @ [Jump]) 1 = jump' n and length (path @ [Jump]) =
Suc s
    by auto
  then show ?case by blast
qed simp

```

```

lemma path-implies-reachable:
  reachable' (length path) (make-moves path 1)
proof (induction path rule: rev-induct)
  case Nil
  then show ?case using Start by auto
next
  case (snoc move path)
  then show ?case
    by (cases move) auto
qed

```

```

lemma heightI[intro]:
  assumes make-moves p 1 = x
  shows height' x ≤ length p
proof -
  from assms have reachable' (length p) x
    using path-implies-reachable by auto
  thus height' x ≤ length p
    unfolding height'-def using arg-min-nat-le[where m=id] by auto
qed

```

```

lemma heightD:
  assumes  $x \neq 0$ 
  obtains p where make-moves p 1 = x and length p = height' x
proof -
  from  $\langle x \neq 0 \rangle$  have reachable' (height' x) x
    using reachable-height by simp
  with that show thesis using path-exists by auto
qed

```

## 0.4 The case for even $k$

From a path leading to  $k$ , we can derive a path to  $k/2$  for each step, look at the change to the current integer without the lowest bit. As it turns out, each of those can be mimicked in one move.

```

fun halfpath :: nat  $\Rightarrow$  path  $\Rightarrow$  path where
halfpath n [] = [] |
halfpath n (Incr # p) =
  (if even n
    then halfpath (Suc n) p
    else Incr # halfpath (Suc n) p) |
halfpath n (Jump # p) =

```

```

    (if even n
      then Jump # halfpath (jump' n) p
      else Incr # halfpath (jump' n) p)

lemma halfpath:
  assumes  $n \neq 0$   $b < 2$ 
    and make-moves  $p$   $(2*n+b) = 2*v$ 
  shows make-moves (halfpath  $(2*n+b)$   $p$ )  $n = v$ 
  using assms
proof (induction p arbitrary: n b)
  case Nil
  then show ?case by simp
next
  case (Cons move p)
  then consider  $b = 0 \mid b = 1$  by fastforce
  thus ?case
  proof cases
    case 1
    then show ?thesis
    proof (cases move)
      case Incr
      from Cons.IH[of n 1]  $\langle n \neq 0 \rangle$  Cons show ?thesis
      unfolding Incr 1 by simp
    next
      case Jump
      from Cons.IH[of jump' n 0]  $\langle n \neq 0 \rangle$  Cons.prem show ?thesis
      unfolding Jump 1 by simp
    qed
  next
    case 2
    then show ?thesis
    proof (cases move)
      case Incr
      from Cons.IH[of Suc n 0]  $\langle n \neq 0 \rangle$  Cons show ?thesis
      unfolding Incr 2 by simp
    next
      case Jump
      from Cons.IH[of Suc n 1]  $\langle n \neq 0 \rangle$  Cons show ?thesis
      unfolding Jump 2 by simp
    qed
  qed
qed

```

```

lemma halfpath-shorter:
  length (halfpath n p)  $\leq$  length p
  apply (induction p arbitrary: n)
  apply simp
  subgoal for move
  apply (cases move)
  by (auto simp add: le-SucI)
done

```

```

lemma halfpath-hd:
   $p \neq [] \implies \text{hd} (\text{halfpath } 1 \text{ } p) = \text{Incr}$ 
  apply (cases p)
  apply simp
  subgoal for move
  apply (cases move)
  by auto
done

```

```

corollary even-reduce:
  assumes  $x \neq 0$ 

```

**shows**  $\text{height}' x < \text{height}' (2*x)$   
**proof** –  
**from** *assms* **have**  $2*x \neq 0$  **by** *simp*  
**then obtain** *path* **where** *path*: *make-moves path 1 = 2\*x*   *length path = height' (2\*x)*  
**using** *heightD* **by** *blast*  
**then have** *path*  $\neq []$  **by** *auto*  
**then obtain** *m p* **where** *m-p: path = m # p*  
**using** *list.exhaust* **by** *blast*  
**let**  $?path' = \text{halfpath} (\text{make-move } m \ 1) \ p$   
**obtain** *b* **where**  $*$ : *make-move m 1 = 2 + b*    $b < 2$   
**by** (*cases m; auto simp add: jump'-def jump-def*)  
**hence** *make-moves p (2 + b) = 2\*x*  
**using** *path m-p* **by** *auto*  
**hence** *make-moves ?path' 1 = x*  
**using** *halfpath[where n=1]* **and**  $*$  **by** *auto*  
**hence**  $\text{height}' x \leq \text{length } ?path'$  **by** *auto*  
**also have**  $\dots \leq \text{length } p$   
**by** (*fact halfpath-shorter*)  
**also have**  $\dots < \text{length } \text{path}$   
**using** *m-p* **by** *auto*  
**also have**  $\dots = \text{height}' (2*x)$   
**by** (*fact path(2)*)  
**finally show**  $\text{height}' x < \text{height}' (2*x)$ .  
**qed**

**corollary** *twopow-reduce*:

**assumes**  $x \neq 0$   
**shows**  $\text{height}' x < \text{height}' (2^\sim(\text{Suc } i) * x)$   
**proof** (*induction i*)  
**case** (*Suc i*)  
**have**  $\text{height}' x < \text{height}' (2^\sim \text{Suc } i * x)$  **by** *fact*  
**also have**  $\dots < \text{height}' (2 * (2^\sim \text{Suc } i * x))$  **using** *assms* **by** (*intro even-reduce*) *auto*  
**finally show**  $?case$  **by** (*simp add: ac-simps*)  
**qed** (*simp add: even-reduce assms*)

## 0.5 The case for odd $k$

The strategy here is similar, but we disregard some high-order bits instead.

**lemma** *jump-multiplicity*:

*multiplicity 2 (jump' x) = multiplicity 2 x*  
**unfolding** *jump'-def jump-def*  
**by** (*auto intro: multiplicity-sum-lt simp del: power-Suc*)

**lemma** *jump-mod*:

$[\text{jump}' (x \bmod 2^\sim n) = \text{jump}' x] \ (\bmod 2^\sim n)$

**proof** (*cases x mod 2^\sim n = x*)

**case** *True*

**then show**  $?thesis$  **by** *simp*

**next**

**case** *False*

**show**  $?thesis$

**proof** (*cases x mod 2^\sim n = 0*)

**case** *True*

**then have**  $2^\sim n \text{ dvd } x$  **by** *auto*

**moreover have**  $x \neq 0$  **using**  $\langle x \bmod 2^\sim n \neq x \rangle \langle x \bmod 2^\sim n = 0 \rangle$  **by** *simp*

**ultimately have**  $\text{multiplicity } 2 \ x \geq n$

**by** (*auto intro: multiplicity-geI*)

**with** *jump-multiplicity* **have**  $\text{multiplicity } 2 \ (\text{jump}' x) \geq n$  **by** *auto*

**hence**  $2^\sim n \text{ dvd } \text{jump}' x$

**by** (*simp add: multiplicity-dvd'*)

**then show**  $?thesis$

```

    using  $\langle x \bmod 2^n = 0 \rangle$  by (auto simp add: cong-def)
next
case False
then have  $x \neq 0$  by (meson mod-0)

have  $x = x \bmod 2^n + 2^n * (x \text{ div } 2^n)$ 
  by simp
also have  $\text{multiplicity } 2 \dots = \text{multiplicity } 2 (x \bmod 2^n)$ 
proof (intro multiplicity-sum-lt)
  show  $x \bmod 2^n \neq 0$  by fact

  have  $x \text{ div } 2^n \neq 0$ 
  proof
    assume  $x \text{ div } 2^n = 0$ 
    then have  $x \bmod 2^n = x$  by presburger
    with  $\langle x \bmod 2^n \neq x \rangle$  show False..
  qed
  thus  $2^n * (x \text{ div } 2^n) \neq 0$  by simp

  have  $\text{multiplicity } 2 (x \bmod 2^n) < n$ 
  using  $\langle x \bmod 2^n \neq 0 \rangle$ 
  by (metis dvd-div-eq-0-iff mod-div-trivial odd-one multiplicity-lessI)

  also have  $n \leq \text{multiplicity } 2 (2^n * (x \text{ div } 2^n))$ 
  using  $\langle 2^n * (x \text{ div } 2^n) \neq 0 \rangle$  dvd-triv-left multiplicity-geI odd-one by blast

  finally show  $\text{multiplicity } 2 (x \bmod 2^n) < \text{multiplicity } 2 (2^n * (x \text{ div } 2^n))$ .
qed
finally have  $[2^n(\text{multiplicity } 2 (x \bmod 2^n) + 1) = 2^n(\text{multiplicity } 2 x + 1)] \pmod{2^n}$ 
  by simp

have  $[\text{jump } (x \bmod 2^n) = \text{jump } x] \pmod{2^n}$ 
  unfolding jump-def by (intro cong-add, simp, fact)
with  $\langle x \neq 0 \rangle$  and  $\langle x \bmod 2^n \neq 0 \rangle$ 
show  $[\text{jump}' (x \bmod 2^n) = \text{jump}' x] \pmod{2^n}$ 
  by (auto simp add: jump'-def)
qed
qed

lemma jump-mod-rollover:
  assumes  $x < 2^n$  and  $\text{jump}' x \geq 2^n$ 
  shows  $[\text{jump}' x = 2^n(\text{multiplicity } 2 x)] \pmod{2^n}$ 
proof -
  have  $x \neq 0$ 
  proof
    assume  $x = 0$ 
    then have  $\text{jump}' x = 0$  by simp
    with assms have  $0 < 2^n$  and  $0 \geq 2^n$  by auto
    thus False by auto
  qed
  have  $x \neq 0$ 
  moreover have  $?v \text{ dvd } x$  by (fact multiplicity-dvd)
  ultimately have  $?v \text{ dvd } \text{jump}' x$  by auto

show ?thesis
proof (cases  $2^n \text{ dvd } ?v$ )
case False
then have  $?v \text{ dvd } 2^n$ 
  by (simp add: dvd-power-iff-le)
then obtain  $m$  where  $m: 2^n = ?v * m$  by auto
moreover from  $\langle ?v \text{ dvd } x \rangle$  obtain  $x'$  where  $x': x = ?v * x'$  by auto

```

moreover from  $\langle ?v \text{ dvd } \text{jump}' x \rangle$  obtain  $j'$  where  $j': \text{jump}' x = ?v * j'$  by *auto*  
ultimately have  $x' < m$  and  $j' \geq m$  using *assms*  
by *auto* (*metis nat-mult-less-cancel-disj*)

moreover have  $j' = x' + 2$

**proof** –

from  $\langle \text{jump}' x = x + 2 * ?v \rangle$  have  $?v * j' = ?v * x' + ?v * 2$   
using  $x' j'$  by *simp*  
also have  $\dots = ?v * (x' + 2)$  by *simp*  
finally show  $j' = x' + 2$   
by (*simp only: mult-cancel1, simp*)

**qed**

ultimately have  $j' = m \vee j' = m + 1$  by *auto*

moreover have  $j' \neq m$

**proof**

assume  $j' = m$   
then have  $2^{\wedge} n = \text{jump}' x$   
using  $j' m$  by *simp*  
hence  $\text{multiplicity } 2 x = \text{multiplicity } (2::\text{nat}) (2^{\wedge} n)$   
by (*simp add: jump-multiplicity*)  
also have  $\dots = n$  by *simp*  
finally have  $?v = 2^{\wedge} n$  by *simp*  
with  $\langle \neg 2^{\wedge} n \text{ dvd } ?v \rangle$  show *False* by *simp*

**qed**

ultimately have  $j' = m + 1$  by *auto*

hence  $\text{jump}' x = 2^{\wedge} n + ?v$  using  $j' m$  by *auto*

thus *?thesis* unfolding *cong-def* by *simp*

**next**

case *True*

with  $\langle ?v \text{ dvd } \text{jump}' x \rangle$  show *?thesis*

unfolding *cong-def* by *auto*

**qed**

**qed**

Instead of constructing the new path explicitly, we inductively prove that one exists. To show that the resulting path is strictly shorter, we maintain the invariant that the path was constructed by removing steps from the original. Thus, since the number we end up on is different, the path cannot be equal and must be shorter.

**inductive** *skipping* :: *path*  $\Rightarrow$  *path*  $\Rightarrow$  *bool* **where**

*SkipEmpty*[*intro*]: *skipping* [] *p* |

*SkipCons*[*intro*]: *skipping p q*  $\Longrightarrow$  *skipping (m # p) (m # q)* |

*SkipSemicons*[*intro*]: *skipping p q*  $\Longrightarrow$  *skipping p (m # q)*

**lemma** *length-skipping*:

assumes *skipping p q*

shows  $\text{length } p \leq \text{length } q$

using *assms* by *induction auto*

**lemma** *skipping-equal*:

assumes *skipping p q* and  $\text{length } p = \text{length } q$

shows  $p = q$

using *assms*

**proof** *induction*

case (*SkipSemicons p q m*)

hence  $\text{length } p \leq \text{length } q$  by (*intro length-skipping*)

hence  $\text{length } p < \text{length } (m \# q)$  by *simp*

with  $\langle \text{length } p = \text{length } (m \# q) \rangle$  have *False* by *simp*

thus *?case..*

**qed** *auto*

```

lemma skipping-self[simp]: skipping a a
  by (induction a) auto

lemma skipping-semiprend[simp]:
  skipping a (p @ a)
  by (induction p) auto

lemma skip-append[intro]:
  assumes skipping p q
  shows skipping (p @ a) (q @ a)
  using assms
  by induction auto

lemma skip-semiappend[intro]:
  assumes skipping p q
  shows skipping p (q @ a)
  using assms
  by induction auto

lemma modpath:
  assumes make-moves p 0 = x
  shows  $\exists p'. \text{make-moves } p' \ 0 = x \bmod 2^k \wedge \text{skipping } p' \ p$ 
  using assms
proof (induction p arbitrary: x k rule: rev-induct)
  case Nil
  then show ?case by force
next
  case (snoc m p)
  then show ?case
  proof (cases x mod 2^k = 0)
    case True
    then show ?thesis
      by (auto intro: exI[of - []])
  next
    case False
    then show ?thesis
    proof (cases m)
      case Incr
      with snoc have x-eq: x = Suc (make-moves p 0) by simp
      with  $\langle x \bmod 2^k \neq 0 \rangle$  have  $\ast: x \bmod 2^k = \text{Suc } (make-moves \ p \ 0 \bmod 2^k)$ 
      using mod-Suc by auto
      from snoc.IH obtain p' where make-moves p' 0 = make-moves p 0 mod 2^k and skipping
p' p
      by auto
      with  $\ast$  have make-moves (p' @ [Incr]) 0 = x mod 2^k
      by auto
      thus ?thesis
      apply (intro exI[of - p' @ [Incr]])
      using  $\langle \text{skipping } p' \ p \rangle \langle m = \text{Incr} \rangle$  by auto
    next
      case Jump
      with snoc have x-eq: x = jump' (make-moves p 0) (is x = jump' ?x') by simp
      hence x-cong: [jump' (?x' mod 2^k) = x] (mod 2^k) by (simp add: jump-mod)
      then consider (nowrap) jump' (?x' mod 2^k) = x mod 2^k
      | (wrap) jump' (?x' mod 2^k) > 2^k
      by (smt  $\langle x \bmod 2^k \neq 0 \rangle$  cong-def linorder-neqE-nat mod-less mod-self)

  then show ?thesis
  proof cases
    case nowrap
    from snoc.IH obtain p' where make-moves p' 0 = ?x' mod 2^k and skipping p' p by
auto

```



```

with nowrap have make-moves ( $p' @ [Jump]$ )  $0 = x \bmod 2^k$  by simp
then show ?thesis
  apply (intro exI[of -  $p' @ [Jump]$ ])
  using ⟨skipping  $p' p$ ⟩  $m = Jump$  by auto
next
  let  $?k' = \text{multiplicity } 2 \ (?x' \bmod 2^k)$ 
  case wrap
  then have jump-mod2k:  $[jump' \ (?x' \bmod 2^k) = 2^{?k}] \ (mod \ 2^k)$ 
    by (intro jump-mod-rollover; auto)
  with x-cong have  $[x = 2^{?k}] \ (mod \ 2^k)$ 
    by (metis cong-trans cong-sym)
  have x-mod:  $x \bmod 2^k = 2^{?k'}$ 
  proof -
    from  $[x = 2^{?k}] \ (mod \ 2^k)$  have  $x \bmod 2^k = 2^{?k'} \bmod 2^k$ 
    unfolding cong-def.
    hence  $x \bmod 2^k = 0 \vee x \bmod 2^k = 2^{?k'}$ 
    by (simp add: exp-mod-exp)
    with  $\langle x \bmod 2^k \neq 0 \rangle$  show  $x \bmod 2^k = 2^{?k'}$ 
    by simp
  qed
  hence  $?k' < k$ 
  by (metis mod-less-divisor nat-power-less-imp-less nat-zero-less-power-iff power-eq-0-iff
power-zero-numeral)
  hence  $Suc \ ?k' \leq k$  by simp
  hence dvd:  $(2::nat)^{Suc \ ?k'} \text{ dvd } 2^k$ 
    using le-imp-power-dvd by blast

  have  $?x' \bmod 2^{Suc \ ?k'} = 2^{?k'}$ 
  proof -
    have  $2^{?k'} \text{ dvd } ?x' \bmod 2^k$  by (fact multiplicity-dvd)
    with  $\langle Suc \ ?k' \leq k \rangle$  have  $2^{?k'} \text{ dvd } ?x'$ 
    by (meson Suc-leD dvd-mod-imp-dvd dvd-power-le gcd-nat.refl)
    hence  $2^{?k'} \text{ dvd } ?x' \bmod 2^{Suc \ ?k'}$ 
    by (simp add: dvd-mod)
    then obtain  $w$  where  $w: ?x' \bmod 2^{Suc \ ?k'} = 2^{?k'} * w$  by blast
    have  $?x' \bmod 2^{Suc \ ?k'} < 2^{Suc \ ?k'}$  by simp
    with  $w$  have  $2^{?k'} * w < 2^{Suc \ ?k'}$  by simp
    hence  $w < 2$  by simp
    hence  $w = 0 \vee w = 1$  by auto
    moreover have  $w \neq 0$ 
    proof
      assume  $w = 0$ 
      with  $w$  have  $2^{Suc \ ?k'} \text{ dvd } ?x'$  by auto
      with dvd have  $2^{Suc \ ?k'} \text{ dvd } ?x' \bmod 2^k$ 
      by (simp add: dvd-mod)
      hence  $?k' \geq Suc \ ?k'$ 
      proof (intro multiplicity-geI)
        show  $?x' \bmod 2^k \neq 0$ 
        by (metis jump'-0 not-less0 wrap)
      qed auto
      thus False by simp
    qed

    ultimately have  $w = 1$  by simp
    with  $w$  show  $?x' \bmod 2^{Suc \ ?k'} = 2^{?k'}$  by simp
  qed
  moreover obtain  $p'$  where make-moves  $p' \ 0 = ?x' \bmod 2^{Suc \ ?k'}$  and skipping  $p' p$ 
    using snoc.IH by blast
  ultimately have make-moves  $p' \ 0 = x \bmod 2^k$  using x-mod by simp
  then show ?thesis
    apply (intro exI[of -  $p'$ ])
    using ⟨skipping  $p' p$ ⟩ by auto
qed

```

```

qed
qed
qed

lemma strip-zero:
  assumes make-moves p 0 = x and x ≠ 0
  shows ∃ p'. make-moves p' 1 = x ∧ length p' < length p
  using assms
proof (induction p)
  case (Cons m p)
  show ?case
  proof (cases m)
    case Incr
    with Cons show ?thesis by (intro exI[of - p]; auto)
  next
    case Jump
    with Cons.premys have make-moves p 0 = x by auto
    with Cons show ?thesis by auto
  qed
qed auto

lemma odd-height:
  assumes odd k and k > 1
  shows height' (2^i) < height' (2^i * k)
proof -
  from assms have 2^i * k ≠ 0
  by (simp add: odd-pos)
  with heightD obtain p where make-moves p 1 = 2^i * k
  and **: length p = height' (2^i * k)
  by blast
  then have p-result: make-moves (Incr # p) 0 = 2^i * k by simp
  with modpath obtain p' where make-moves p' 0 = 2^i * k mod 2^Suc i
  and skipping: skipping p' (Incr # p)
  by blast
  hence p'-result: make-moves p' 0 = 2^i
  using ⟨odd k⟩
  by (simp add: odd-iff-mod-2-eq-one)
  from skipping and length-skipping have length p' ≤ Suc (length p)
  by force
  also have length p' ≠ Suc (length p)
  proof
    assume length p' = Suc (length p)
    hence p' = Incr # p by (intro skipping-equal; auto simp add: skipping)
    hence make-moves p' 0 = make-moves (Incr # p) 0 by simp
    hence k = 1 using p-result p'-result by simp
    with ⟨k > 1⟩ show False by simp
  qed
  finally have *: length p' ≤ length p by simp

  have (2::nat)^i ≠ 0 by simp
  with p'-result obtain p''
  where p'': make-moves p'' 1 = 2^i
  and length p'' < length p'
  using strip-zero by blast
  with * have length p'' < length p by simp
  moreover from p'' have height' (2^i) ≤ length p'' by auto
  ultimately show height' (2^i) < height' (2^i * k) using ** by simp
qed

theorem problem5:
  assumes k ≥ 2
  shows height (2^i) < height (2^i * k)
proof -

```

```

let ?n = multiplicity 2 k
obtain k' where k = 2?n * k' and odd k'
  using multiplicity-decompose' assms
  by (metis not-numeral-le-zero odd-one)
then have height' (2i) < height' (2i * k)
proof (cases ?n)
  case 0
  hence k = k' using ⟨k = 2?n * k'⟩ by simp
  hence odd k using ⟨odd k'⟩ by simp
  thus height' (2i) < height' (2i * k)
    using assms by (auto intro: odd-height)
next
  case (Suc n)
  have height' (2i) < height' (2Suc n * 2i)
    by (intro twopow-reduce; auto)
  also have ... = height' (2?n + i)
    by (simp add: power-add Suc ac-simps)
  also have ... ≤ height' (2?n + i * k')
  proof (cases k' = 1)
    case True
    then show ?thesis by simp
  next
    case False
    then have height' (2?n + i) < height' (2?n + i * k')
      using ⟨odd k'⟩ by (intro odd-height; auto simp add: Suc-lessI odd-pos)
    then show ?thesis by simp
  qed
  also have ... = height' (2i * (2?n * k'))
    by (simp add: ac-simps power-add)
  finally show height' (2i) < height' (2i * k)
    using ⟨k = 2?n * k'⟩ by simp
  qed
  thus ?thesis by simp
qed
end

```