

Formalization of "Knight's Tour Revisited"

Lukas Koller

December 30, 2021

Contents

1	Definitions	1
2	Executable Checker for a Knight's Path	2
2.1	Implementation of an Executable Checker	2
2.2	Correctness Proof of the Executable Checker	3
3	Proof for some Properties of <i>knights-path</i> and <i>knights-circuit</i>	5
4	Transpose and Mirror Paths and Boards	13
4.1	Transpose Paths and Boards	13
4.2	Mirror Paths and Boards	16
5	Path and Board Translation	25
5.1	Implementation of Path and Board Translation	25
5.2	Correctness Proofs for Path and Board Translation	25
6	Path Parser and Path Construction	35
7	Paths for $5 \times m$-Boards	36
8	Paths and Circuits for $6 \times m$-Boards	43
9	Paths and Circuits for $8 \times m$-Boards	51
10	Paths and Circuits for $n \times m$-Boards	61

```
theory KnightsTour
  imports Main
begin
```

This is a formalization of [?]. In [?] the existence of Knight's paths and Knight's circuits are proved for arbitrary $n \times m$ -boards with $\min n m \geq 5$.

A Knight's path is an instance of the Hamiltonian Path Problem. A Knight's path is a sequence of squares on a chessboard s.t. every step in sequence is a valid move for a Knight. A Knight is a chess figure that is only able to move two squares vertically and one square horizontally or two squares horizontally and one square vertically. A Knight's circuit is a Knight's path, where additionally the Knight can move from the last square to the first square of the path, forming a loop.

The main idea for the proof of the existence of a Knight's path is to inductively construct paths from a few pre-computed paths for small boards, e.g. 5×5 , 5×6 , ..., 8×9 . The paths for small boards are transformed (i.e. transpose, mirror, translate) and combined to create paths for larger boards.

While formalizing the proofs I have noticed two mistakes in the original proof by Cull and De Curtins: (i) the pre-computed path for the 6×6 board that ends in the upper-left (in Figure 2) and (ii) the pre-computed path for the 8×8 board that ends in the upper-left (in Figure 5) are false. I.e. on the 6×6 board the Knight cannot step from square 26 to square 27; in the 8×8 board the Knight cannot step from square 27 to square 28.

1 Definitions

type-synonym $\text{square} = \text{int} \times \text{int}$

type-synonym $\text{board} = \text{square set}$

A $(n \times m)$ -board is the set of all squares (i, j) where $1 \leq i \leq n$ and $1 \leq j \leq m$. $(1, 1)$ is the lower-left corner.

definition $\text{board} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{board}$ **where**

$\text{board } n \ m = \{(i, j) \mid i \ j. \ 1 \leq i \wedge i \leq \text{int } n \wedge 1 \leq j \wedge j \leq \text{int } m\}$

A path is a sequence of steps on a board. A path is represented by the list of visited squares on the board. Each square on the $(n \times m)$ -board is identified by its coordinates (i, j) .

type-synonym $\text{path} = \text{square list}$

Predicate that characterizes a valid step. A Knight can only move two squares vertically and one square horizontally or two squares horizontally and one square vertically. Therefore, a knight at position (i, j) can only move to $(i \pm 1, j \pm 2)$ or $(i \pm 2, j \pm 1)$.

definition $\text{valid-step} :: \text{square} \Rightarrow \text{square} \Rightarrow \text{bool}$ **where**

$\text{valid-step } s_i \ s_j \equiv (\text{case } s_i \text{ of } (i, j) \Rightarrow s_j \in \{(i+1, j+2), (i-1, j+2), (i+1, j-2), (i-1, j-2), (i+2, j+1), (i-2, j+1), (i+2, j-1), (i-2, j-1)\})$

Now we define an inductive predicate that characterizes a Knight's path. The knight can only make a valid step to a position on the board that has not been visited yet.

inductive *knight's-path* :: *board* \Rightarrow *path* \Rightarrow *bool* **where**
 knight's-path $\{s_i\} [s_i]$
 | $s_i \notin b \Rightarrow \text{valid-step } s_i \ s_j \Rightarrow \text{knight's-path } b \ (s_j \# ps) \Rightarrow \text{knight's-path } (b \cup \{s_i\}) \ (s_i \# s_j \# ps)$

code-pred *knight's-path* .

A sequence is a Knight's circuit iff the sequence is a Knight's path and there is a valid step from the last square to the first square.

definition *knight's-circuit* $b \ ps \equiv (\text{knight's-path } b \ ps \wedge \text{valid-step } (\text{last } ps) \ (\text{hd } ps))$

2 Executable Checker for a Knight's Path

This section gives the implementation and correctness-proof for an executable checker for a knight's-path wrt. the definition *knight's-path*.

2.1 Implementation of an Executable Checker

fun *row-exec* :: *nat* \Rightarrow *int set* **where**
 row-exec 0 = {}
 | *row-exec* $m = \text{insert } (\text{int } m) \ (\text{row-exec } (m-1))$

fun *board-exec-aux* :: *nat* \Rightarrow *int set* \Rightarrow *board* **where**
 board-exec-aux 0 $M = \{\}$
 | *board-exec-aux* $k \ M = \{(\text{int } k, j) \mid j. j \in M\} \cup \text{board-exec-aux } (k-1) \ M$

Compute a board.

fun *board-exec* :: *nat* \Rightarrow *nat* \Rightarrow *board* **where**
 board-exec $n \ m = \text{board-exec-aux } n \ (\text{row-exec } m)$

fun *step-checker* :: *square* \Rightarrow *square* \Rightarrow *bool* **where**
 step-checker $(i, j) \ (i', j') =$
 $((i+1, j+2) = (i', j') \vee (i-1, j+2) = (i', j') \vee (i+1, j-2) = (i', j') \vee (i-1, j-2) = (i', j'))$
 $\vee ((i+2, j+1) = (i', j') \vee (i-2, j+1) = (i', j') \vee (i+2, j-1) = (i', j') \vee (i-2, j-1) = (i', j'))$

fun *path-checker* :: *board* \Rightarrow *path* \Rightarrow *bool* **where**
 path-checker $b \ [] = \text{False}$
 | *path-checker* $b \ [s_i] = (\{s_i\} = b)$
 | *path-checker* $b \ (s_i \# s_j \# ps) = (s_i \in b \wedge \text{step-checker } s_i \ s_j \wedge \text{path-checker } (b - \{s_i\}) \ (s_j \# ps))$

fun *circuit-checker* :: *board* \Rightarrow *path* \Rightarrow *bool* **where**
 circuit-checker $b \ ps = (\text{path-checker } b \ ps \wedge \text{step-checker } (\text{last } ps) \ (\text{hd } ps))$

2.2 Correctness Proof of the Executable Checker

lemma *row-exec-leq*: $j \in \text{row-exec } m \longleftrightarrow 1 \leq j \wedge j \leq \text{int } m$
by (*induction m*) *auto*

lemma *board-exec-aux-leq-mem*: $(i,j) \in \text{board-exec-aux } k \ M \longleftrightarrow 1 \leq i \wedge i \leq \text{int } k \wedge j \in M$
by (*induction k M rule: board-exec-aux.induct*) *auto*

lemma *board-exec-leq*: $(i,j) \in \text{board-exec } n \ m \longleftrightarrow 1 \leq i \wedge i \leq \text{int } n \wedge 1 \leq j \wedge j \leq \text{int } m$
using *board-exec-aux-leq-mem row-exec-leq* **by** *auto*

lemma *board-exec-correct*: $\text{board } n \ m = \text{board-exec } n \ m$
unfolding *board-def* **using** *board-exec-leq* **by** *auto*

lemma *step-checker-correct*: $\text{step-checker } s_i \ s_j \longleftrightarrow \text{valid-step } s_i \ s_j$

proof

assume *step-checker* $s_i \ s_j$
then show *valid-step* $s_i \ s_j$
unfolding *valid-step-def*
apply (*cases* s_i)
apply (*cases* s_j)
apply *auto*
done

next

assume *assms*: *valid-step* $s_i \ s_j$
then show *step-checker* $s_i \ s_j$
unfolding *valid-step-def* **by** *auto*

qed

lemma *step-checker-rev*: $\text{step-checker } (i,j) \ (i',j') \implies \text{step-checker } (i',j') \ (i,j)$
apply (*simp only: step-checker.simps*)
by (*elim disjE*) *auto*

lemma *knight-path-intro-rev*:

assumes $s_i \in b \ \text{valid-step } s_i \ s_j \ \text{knight-path } (b - \{s_i\}) \ (s_j \# ps)$
shows $\text{knight-path } b \ (s_i \# s_j \# ps)$
using *assms*

proof –

assume *assms*: $s_i \in b \ \text{valid-step } s_i \ s_j \ \text{knight-path } (b - \{s_i\}) \ (s_j \# ps)$
then have $s_i \notin (b - \{s_i\}) \ b - \{s_i\} \cup \{s_i\} = b$
by *auto*
then show *?thesis*
using *assms knight-path.intros(2)[of s_i b - {s_i}]* **by** *auto*

qed

Final correctness corollary for the executable checker *path-checker*.

lemma *path-checker-correct*: $\text{path-checker } b \ ps \longleftrightarrow \text{knight-path } b \ ps$

proof

```

assume path-checker b ps
then show knights-path b ps
proof (induction rule: path-checker.induct)
  case ( $\exists s_i s_j xs b$ )
    then show ?case using step-checker-correct knights-path-intro-rev by auto
  qed (auto intro: knights-path.intros)
next
  assume knights-path b ps
  then show path-checker b ps
    using step-checker-correct
    by (induction rule: knights-path.induct) (auto elim: knights-path.cases)
qed

corollary knights-path-exec-simp: knights-path (board n m) ps  $\longleftrightarrow$  path-checker
(board-exec n m) ps
  using board-exec-correct path-checker-correct[symmetric] by simp

lemma circuit-checker-correct: circuit-checker b ps  $\longleftrightarrow$  knights-circuit b ps
  unfolding knights-circuit-def using path-checker-correct step-checker-correct by
auto

corollary knights-circuit-exec-simp:
  knights-circuit (board n m) ps  $\longleftrightarrow$  circuit-checker (board-exec n m) ps
  using board-exec-correct circuit-checker-correct[symmetric] by simp

```

3 Proof for some Properties of *knights-path* and *knights-circuit*

```

lemma board-leg-subset:  $n_1 \leq n_2 \wedge m_1 \leq m_2 \implies \text{board } n_1 \ m_1 \subseteq \text{board } n_2 \ m_2$ 
  unfolding board-def by auto

lemma finite-row-exec: finite (row-exec m)
  by (induction m) auto

lemma finite-board-exec-aux: finite M  $\implies$  finite (board-exec-aux n M)
  by (induction n) auto

lemma board-finite: finite (board n m)
  using finite-board-exec-aux finite-row-exec by (simp only: board-exec-correct) auto

lemma card-row-exec: card (row-exec m) = m
proof (induction m)
  case (Suc m)
    have int (Suc m)  $\notin$  row-exec m
      using row-exec-leg by auto
    then have card (insert (int (Suc m)) (row-exec m)) = 1 + card (row-exec m)
      using card-Suc-eq by (metis Suc plus-1-eq-Suc row-exec.simps(1))
    then have card (row-exec (Suc m)) = 1 + card (row-exec m)
      by auto
    then show ?case using Suc.IH by auto

```

qed *auto*

lemma *set-comp-ins*:

$\{(k,j) \mid j. j \in \text{insert } x \ M\} = \text{insert } (k,x) \ \{(k,j) \mid j. j \in M\}$ (**is** $?Mi = ?iM$)

proof

show $?Mi \subseteq ?iM$

proof

fix y **assume** $y \in ?Mi$

then obtain j **where** $[simp]: y = (k,j)$ **and** $j \in \text{insert } x \ M$ **by** *blast*

then have $j = x \vee j \in M$ **by** *auto*

then show $y \in ?iM$ **by** (*elim disjE*) *auto*

qed

next

show $?iM \subseteq ?Mi$

proof

fix y **assume** $y \in ?iM$

then obtain j **where** $[simp]: y = (k,j)$ **and** $j \in \text{insert } x \ M$ **by** *blast*

then have $j = x \vee j \in M$ **by** *auto*

then show $y \in ?Mi$ **by** (*elim disjE*) *auto*

qed

qed

lemma *finite-card-set-comp*: $\text{finite } M \implies \text{card } \{(k,j) \mid j. j \in M\} = \text{card } M$

proof (*induction M rule: finite-induct*)

case (*insert x M*)

then show $?case$ **using** *set-comp-ins*[*of k x M*] **by** *auto*

qed *auto*

lemma *card-board-exec-aux*: $\text{finite } M \implies \text{card } (\text{board-exec-aux } k \ M) = k * \text{card } M$

proof (*induction k*)

case (*Suc k*)

let $?M' = \{(int \ (Suc \ k), j) \mid j. j \in M\}$

let $?rec-k = \text{board-exec-aux } k \ M$

have *finite*: $\text{finite } ?M' \ \text{finite } ?rec-k$

using *Suc finite-board-exec-aux* **by** *auto*

then have *card-Un-simp*: $\text{card } (?M' \cup ?rec-k) = \text{card } ?M' + \text{card } ?rec-k$

using *board-exec-aux-leq-mem card-Un-Int*[*of ?M' ?rec-k*] **by** *auto*

have *card-M*: $\text{card } ?M' = \text{card } M$

using *Suc finite-card-set-comp* **by** *auto*

have *card* (*board-exec-aux* (*Suc k*) *M*) = $\text{card } ?M' + \text{card } ?rec-k$

using *card-Un-simp* **by** *auto*

also have $\dots = \text{card } M + k * \text{card } M$

using *Suc card-M* **by** *auto*

also have $\dots = (Suc \ k) * \text{card } M$

by *auto*

finally show $?case$.

qed *auto*

lemma *card-board*: $\text{card } (\text{board } n \ m) = n * m$

proof –

have $\text{card } (\text{board } n \ m) = \text{card } (\text{board-exec-aux } n \ (\text{row-exec } m))$

using *board-exec-correct* **by** *auto*

also have $\dots = n * m$

using *card-row-exec card-board-exec-aux finite-row-exec* **by** *auto*

finally show *?thesis* .

qed

lemma *knight-path-board-non-empty*: $\text{knight-path } b \ ps \implies b \neq \{\}$

by (*induction arbitrary: ps rule: knight-path.induct*) *auto*

lemma *knight-path-board-m-n-geq-1*: $\text{knight-path } (\text{board } n \ m) \ ps \implies \min n \ m \geq 1$

unfolding *board-def* **using** *knight-path-board-non-empty* **by** *fastforce*

lemma *knight-path-non-nil*: $\text{knight-path } b \ ps \implies ps \neq []$

by (*induction arbitrary: b rule: knight-path.induct*) *auto*

lemma *knight-path-set-eq*: $\text{knight-path } b \ ps \implies \text{set } ps = b$

by (*induction rule: knight-path.induct*) *auto*

lemma *knight-path-subset*:

$\text{knight-path } b_1 \ ps_1 \implies \text{knight-path } b_2 \ ps_2 \implies \text{set } ps_1 \subseteq \text{set } ps_2 \iff b_1 \subseteq b_2$

using *knight-path-set-eq* **by** *auto*

lemma *knight-path-board-unique*: $\text{knight-path } b_1 \ ps \implies \text{knight-path } b_2 \ ps \implies b_1 = b_2$

using *knight-path-set-eq* **by** *auto*

lemma *valid-step-neq*: $\text{valid-step } s_i \ s_j \implies s_i \neq s_j$

unfolding *valid-step-def* **by** *auto*

lemma *valid-step-non-transitive*: $\text{valid-step } s_i \ s_j \implies \text{valid-step } s_j \ s_k \implies \neg \text{valid-step } s_i \ s_k$

proof –

assume *assms*: $\text{valid-step } s_i \ s_j \ \text{valid-step } s_j \ s_k$

obtain $i_i \ j_i \ i_j \ j_j \ i_k \ j_k$ **where** [*simp*]: $s_i = (i_i, j_i) \ s_j = (i_j, j_j) \ s_k = (i_k, j_k)$ **by** *force*

then have $\text{step-checker } (i_i, j_i) \ (i_j, j_j) \ \text{step-checker } (i_j, j_j) \ (i_k, j_k)$

using *assms step-checker-correct* **by** *auto*

then show $\neg \text{valid-step } s_i \ s_k$

apply (*simp add: step-checker-correct[symmetric]*)

apply (*elim disjE*)

apply *auto*

done

qed

```

lemma knights-path-distinct: knights-path b ps  $\implies$  distinct ps
proof (induction rule: knights-path.induct)
  case (2 si b sj ps)
  then have  $s_i \notin \text{set } (s_j \# ps)$ 
    using knights-path-set-eq valid-step-neq by blast
  then show ?case using 2 by auto
qed auto

lemma knights-path-length: knights-path b ps  $\implies$  length ps = card b
  using knights-path-set-eq knights-path-distinct by (metis distinct-card)

lemma knights-path-take:
  assumes knights-path b ps  $0 < k$   $k < \text{length } ps$ 
  shows knights-path (set (take k ps)) (take k ps)
  using assms
proof (induction arbitrary: k rule: knights-path.induct)
  case (2 si b sj ps)
  then have  $k = 1 \vee k = 2 \vee 2 < k$  by force
  then show ?case
    using 2
  proof (elim disjE)
    assume  $k = 2$ 
    then have  $\text{take } k (s_i \# s_j \# ps) = [s_i, s_j]$   $s_i \notin \{s_j\}$  using 2 valid-step-neq by
auto
    then show ?thesis using 2 knights-path.intros by auto
  next
    assume  $2 < k$ 
    then have k-simps:  $k-2 = k-1-1$   $0 < k-2$   $k-2 < \text{length } ps$  and
      take-simp1:  $\text{take } k (s_i \# s_j \# ps) = s_i \# \text{take } (k-1) (s_j \# ps)$  and
      take-simp2:  $\text{take } k (s_i \# s_j \# ps) = s_i \# s_j \# \text{take } (k-1-1) ps$ 
    using assms 2 take-Cons'[of k si sj # ps] take-Cons'[of k-1 sj ps] by auto
    then have knights-path (set (take (k-1) (sj # ps))) (take (k-1) (sj # ps))
      using 2 k-simps by auto
    then have kp: knights-path (set (take (k-1) (sj # ps))) (sj # take (k-2) ps)
      using take-Cons'[of k-1 sj ps] by (auto simp: k-simps elim: knights-path.cases)

    have no-mem:  $s_i \notin \text{set } (\text{take } (k-1) (s_j \# ps))$ 
      using 2 set-take-subset[of k-1 sj # ps] knights-path-set-eq by blast
    have knights-path (set (take (k-1) (sj # ps)))  $\cup \{s_i\}$  (si # sj # take (k-2) ps)
      using knights-path.intros(2)[OF no-mem <valid-step si sj> kp] by auto
    then show ?thesis using k-simps take-simp2 knights-path-set-eq by metis
  qed (auto intro: knights-path.intros)
qed auto

lemma knights-path-drop:
  assumes knights-path b ps  $0 < k$   $k < \text{length } ps$ 
  shows knights-path (set (drop k ps)) (drop k ps)
  using assms

```



```

proof (induction arbitrary: k rule: knights-path.induct)
  case (2 si b sj ps)
  then have (k = 1 ∧ ps = []) ∨ (k = 1 ∧ ps ≠ []) ∨ 1 < k by force
  then show ?case
    using 2
  proof (elim disjE)
    assume k = 1 ∧ ps ≠ []
    then show ?thesis using 2 knights-path-set-eq by force
  next
    assume 1 < k
    then have 0 < k-1 k-1 < length (sj#ps) drop k (si#sj#ps) = drop (k-1)
      (sj#ps)
    using assms 2 drop-Cons'[of k si sj#ps] by auto
    then show ?thesis
      using 2 by auto
  qed (auto intro: knights-path.intros)
qed auto

```

A Knight's path can be split to form two new disjoint Knight's paths.

corollary *knights-path-split*:

```

assumes knights-path b ps 0 < k k < length ps
shows
  ∃ b1 b2. knights-path b1 (take k ps) ∧ knights-path b2 (drop k ps) ∧ b1 ∪ b2 = b
  ∧ b1 ∩ b2 = {}
using assms
proof -
  let ?b1=set (take k ps)
  let ?b2=set (drop k ps)
  have kp1: knights-path ?b1 (take k ps) and kp2: knights-path ?b2 (drop k ps)
    using assms knights-path-take knights-path-drop by auto
  have union: ?b1 ∪ ?b2 = b
    using assms knights-path-set-eq by (metis append-take-drop-id set-append)
  have inter: ?b1 ∩ ?b2 = {}
    using assms knights-path-distinct by (metis append-take-drop-id distinct-append)
  show ?thesis using kp1 kp2 union inter by auto
qed

```

Append two disjoint Knight's paths.

corollary *knights-path-append*:

```

assumes knights-path b1 ps1 knights-path b2 ps2 b1 ∩ b2 = {} valid-step (last
ps1) (hd ps2)
shows knights-path (b1 ∪ b2) (ps1 @ ps2)
using assms
proof (induction arbitrary: ps2 b2 rule: knights-path.induct)
  case (1 si)
  then have si ∉ b2 ps2 ≠ [] valid-step si (hd ps2) knights-path b2 (hd ps2#tl ps2)

    using knights-path-non-nil by auto
  then have knights-path (b2 ∪ {si}) (si#hd ps2#tl ps2)

```

```

    using knights-path.intros by blast
    then show ?case using ⟨ps2 ≠ []⟩ by auto
next
  case (2 si b1 sj ps1)
  then have si ∉ b1 ∪ b2 valid-step si sj knights-path (b1 ∪ b2) (sj#ps1@ps2) by
auto
  then have knights-path (b1 ∪ b2 ∪ {si}) (si#sj#ps1@ps2)
    using knights-path.intros by auto
  then show ?case by auto
qed

```

lemma *valid-step-rev*: $\text{valid-step } s_i s_j \implies \text{valid-step } s_j s_i$
 using *step-checker-correct step-checker-rev* by (metis *prod.exhaust-sel*)

Reverse a Knight's path.

corollary *knights-path-rev*:
 assumes *knights-path* b ps
 shows *knights-path* b (rev ps)
 using *assms*
proof (*induction rule: knights-path.induct*)
 case (2 s_i b s_j ps)
 then have *knights-path* {s_i} [s_i] b ∩ {s_i} = {} *valid-step* (last (rev (s_j # ps)))
 (hd [s_i])
 using *valid-step-rev* by (auto intro: *knights-path.intros*)
 then have *knights-path* (b ∪ {s_i}) ((rev (s_j#ps))@[s_i])
 using 2 *knights-path-append* by blast
 then show ?case by auto
qed (auto intro: *knights-path.intros*)

Reverse Knight's path.

corollary *knights-circuit-rev*:
 assumes *knights-circuit* b ps
 shows *knights-circuit* b (rev ps)
 using *assms knights-path-rev valid-step-rev*
 unfolding *knights-circuit-def* by (auto simp: *hd-rev last-rev*)

lemma *knights-circuit-rotate1*:
 assumes *knights-circuit* b (s_i#ps)
 shows *knights-circuit* b (ps@[s_i])
proof (*cases ps = []*)
 case *True*
 then show ?thesis using *assms* by auto
next
 case *False*
 have kp1: *knights-path* b (s_i#ps) *valid-step* (last (s_i#ps)) (hd (s_i#ps))
 using *assms unfolding knights-circuit-def* by auto

```

then have kp-elim:  $s_i \notin (b - \{s_i\})$  valid-step  $s_i$  (hd ps) knight-path ( $b - \{s_i\}$ )
ps
  using  $\langle ps \neq [] \rangle$  by (auto elim: knight-path.cases)
then have vs': valid-step (last (ps@[s_i])) (hd (ps@[s_i]))
  using  $\langle ps \neq [] \rangle$  valid-step-rev by auto

have kp2: knight-path  $\{s_i\}$  [s_i] ( $b - \{s_i\}) \cap \{s_i\} = \{\}$ 
  by (auto intro: knight-path.intros)

have vs: valid-step (last ps) (hd [s_i])
  using  $\langle ps \neq [] \rangle$   $\langle \text{valid-step} (\text{last } (s_i \# ps)) (\text{hd } (s_i \# ps)) \rangle$  by auto

have ( $b - \{s_i\} \cup \{s_i\} = b$ )
  using kp1 kp-elim knight-path-set-eq by force
then show ?thesis
  unfolding knight-circuit-def
  using vs knight-path-append[OF  $\langle \text{knight-path } (b - \{s_i\}) ps \rangle$  kp2] vs' by auto
qed

```

A Knight's circuit can be rotated to start at any square on the board.

```

lemma knight-circuit-rotate-to:
  assumes knight-circuit b ps hd (drop k ps) =  $s_i$   $k < \text{length } ps$ 
  shows  $\exists ps'. \text{knight-circuit } b ps' \wedge \text{hd } ps' = s_i$ 
  using assms
proof (induction k arbitrary: b ps)
  case (Suc k)
  let  $?s_j = \text{hd } ps$ 
  let  $?ps' = \text{tl } ps$ 
  show ?case
  proof (cases  $s_i = ?s_j$ )
  case True
  then show ?thesis using Suc by auto
next
  case False
  then have  $?ps' \neq []$ 
  using Suc by (metis drop-Nil drop-Suc drop-eq-Nil2 le-antisym nat-less-le)
  then have knight-circuit b ( $?s_j \# ?ps'$ )
  using Suc by (metis list.exhaust-sel tl-Nil)
  then have knight-circuit b ( $?ps' @ [?s_j]$ ) hd (drop k ( $?ps' @ [?s_j]$ )) =  $s_i$ 
  using Suc knight-circuit-rotate1 by (auto simp: drop-Suc)
  then show ?thesis using Suc by auto
qed
qed auto

```

For positive boards (1,1) can only have (2,3) and (3,2) as a neighbour.

```

lemma valid-step-1-1:
  assumes valid-step (1,1) (i,j)  $i > 0$   $j > 0$ 
  shows  $(i,j) = (2,3) \vee (i,j) = (3,2)$ 
  using assms unfolding valid-step-def by auto

```

```

lemma list-len-g-1-split:  $\text{length } xs > 1 \implies \exists x_1 \ x_2 \ xs'. \ xs = x_1 \# x_2 \# xs'$ 
proof (induction xs)
  case (Cons x xs)
  then have  $\text{length } xs > 0$  by auto
  then have  $\text{length } xs \geq 1$  by presburger
  then have  $\text{length } xs = 1 \vee \text{length } xs > 1$  by auto
  then show ?case
  proof (elim disjE)
    assume  $\text{length } xs = 1$ 
    then obtain  $x_1$  where  $[simp]: xs = [x_1]$ 
    using length-Suc-conv[of xs 0] by auto
    then show ?thesis by auto
  next
    assume  $1 < \text{length } xs$ 
    then show ?thesis using Cons by auto
  qed
qed auto

lemma list-len-g-3-split:  $\text{length } xs > 3 \implies \exists x_1 \ x_2 \ xs' \ x_3. \ xs = x_1 \# x_2 \# xs' @ [x_3]$ 
proof (induction xs)
  case (Cons x xs)
  then have  $\text{length } xs = 3 \vee \text{length } xs > 3$  by auto
  then show ?case
  proof (elim disjE)
    assume  $\text{length } xs = 3$ 
    then obtain  $x_1 \ xs_1$  where  $[simp]: xs = x_1 \# xs_1 \ \text{length } xs_1 = 2$ 
    using length-Suc-conv[of xs 2] by auto
    then obtain  $x_2 \ xs_2$  where  $[simp]: xs_1 = x_2 \# xs_2 \ \text{length } xs_2 = 1$ 
    using length-Suc-conv[of xs_1 1] by auto
    then obtain  $x_3$  where  $[simp]: xs_2 = [x_3]$ 
    using length-Suc-conv[of xs_2 0] by auto
    then show ?thesis by auto
  next
    assume  $\text{length } xs > 3$ 
    then show ?thesis using Cons by auto
  qed
qed auto

```

Any Knight's circuit on a positive board can be rotated to start with (1,1) and end with (3,2).

corollary *rotate-knights-circuit*:

```

assumes knights-circuit (board n m)  $ps \ \text{min } n \ m \geq 5$ 
shows  $\exists ps. \ \text{knights-circuit} \ (\text{board } n \ m) \ ps \wedge \text{hd } ps = (1,1) \wedge \text{last } ps = (3,2)$ 
using assms
proof –
  let  $?b = \text{board } n \ m$ 
  have knights-path  $?b \ ps$ 
  using assms unfolding knights-circuit-def by auto

```

```

then have  $(1,1) \in \text{set } ps$ 
  using assms knights-path-set-eq by (auto simp: board-def)
then obtain  $k$  where  $\text{hd } (\text{drop } k \text{ } ps) = (1,1)$   $k < \text{length } ps$ 
  by (metis hd-drop-conv-nth in-set-conv-nth)
then obtain  $ps_r$  where  $ps_r\text{-prems}: \text{knights-circuit } ?b \text{ } ps_r \text{ } \text{hd } ps_r = (1,1)$ 
  using assms knights-circuit-rotate-to by blast
then have  $kp: \text{knights-path } ?b \text{ } ps_r$  and  $\text{valid-step } (\text{last } ps_r) (1,1)$ 
  unfolding knights-circuit-def by auto

have  $(1,1) \in ?b (1,2) \in ?b (1,3) \in ?b$ 
  using assms unfolding board-def by auto
then have  $(1,1) \in \text{set } ps_r (1,2) \in \text{set } ps_r (1,3) \in \text{set } ps_r$ 
  using  $kp$  knights-path-set-eq by auto

have  $3 < \text{card } ?b$ 
  using assms board-leq-subset card-board[of 5 5]
    card-mono[OF board-finite[of  $n \ m$ ], of board 5 5] by auto
then have  $3 < \text{length } ps_r$ 
  using knights-path-length kp by auto
then obtain  $s_j \text{ } ps' \text{ } s_k$  where  $[simp]: ps_r = (1,1) \# s_j \# ps'@[s_k]$ 
  using  $\langle \text{hd } ps_r = (1,1) \rangle$  list-len-g-3-split[of  $ps_r$ ] by auto
have  $s_j \neq s_k$ 
  using  $kp$  knights-path-distinct by force

have  $vs\text{-}s_k: \text{valid-step } s_k (1,1)$ 
  using  $\langle \text{valid-step } (\text{last } ps_r) (1,1) \rangle$  by simp

have  $vs\text{-}s_j: \text{valid-step } (1,1) s_j$  and  $kp': \text{knights-path } (?b - \{(1,1)\}) (s_j \# ps'@[s_k])$ 
  using  $kp$  by (auto elim: knights-path.cases)

have  $s_j \in \text{set } ps_r \text{ } s_k \in \text{set } ps_r$  by auto
then have  $s_j \in ?b \text{ } s_k \in ?b$ 
  using  $kp$  knights-path-set-eq by blast+
then have  $0 < \text{fst } s_j \wedge 0 < \text{snd } s_j \text{ } 0 < \text{fst } s_k \wedge 0 < \text{snd } s_k$ 
  unfolding board-def by auto
then have  $s_k = (2,3) \vee s_k = (3,2) \text{ } s_j = (2,3) \vee s_j = (3,2)$ 
  using  $vs\text{-}s_k \text{ } vs\text{-}s_j \text{ } \text{valid-step-1-1} \text{ } \text{valid-step-rev}$  by (metis prod.collapse)+
then have  $s_k = (3,2) \vee s_j = (3,2)$ 
  using  $\langle s_j \neq s_k \rangle$  by auto
then show ?thesis
proof (elim disjE)
  assume  $s_k = (3,2)$ 
  then have  $\text{last } ps_r = (3,2)$  by auto
  then show ?thesis using  $ps_r\text{-prems}$  by auto
next
  assume  $s_j = (3,2)$ 
  then have  $vs: \text{valid-step } (\text{last } ((1,1) \# \text{rev } (s_j \# ps'@[s_k]))) (\text{hd } ((1,1) \# \text{rev } (s_j \# ps'@[s_k])))$ 
    unfolding valid-step-def by auto

```

```

have rev-simp: rev (sj#ps'@[sk]) = sk#(rev ps')@[sj] by auto

have knights-path (?b - {(1,1)}) (rev (sj#ps'@[sk]))
  using knights-path-rev[OF kp] by auto
then have (1,1) ∉ (?b - {(1,1)}) valid-step (1,1) sk
  knights-path (?b - {(1,1)}) (sk#(rev ps')@[sj])
  using assms vs-sk valid-step-rev by (auto simp: rev-simp)
then have knights-path (?b - {(1,1)} ∪ {(1,1)}) ((1,1)#sk#(rev ps')@[sj])
  using knights-path.intros(2)[of (1,1) ?b - {(1,1)} sk (rev ps')@[sj]] by auto
then have knights-path ?b ((1,1)#rev (sj#ps'@[sk]))
  using assms by (simp add: board-def insert-absorb rev-simp)
then have knights-circuit ?b ((1,1)#rev (sj#ps'@[sk]))
  unfolding knights-circuit-def using vs by auto
then show ?thesis
  using ⟨sj = (3,2)⟩ by auto
qed
qed

```

4 Transpose and Mirror Paths and Boards

4.1 Transpose Paths and Boards

definition *transpose-square* $s_i = (\text{case } s_i \text{ of } (i,j) \Rightarrow (j,i))$

```

fun transpose :: path ⇒ path where
  transpose [] = []
| transpose (si#ps) = (transpose-square si)#transpose ps

```

definition *transpose-board* :: board ⇒ board **where**

transpose-board $b \equiv \{(j,i) \mid i.j. (i,j) \in b\}$

lemma *transpose2*: *transpose-square* (*transpose-square* s_i) = s_i
unfolding *transpose-square-def* **by** (auto split: prod.splits)

lemma *transpose-nil*: $ps = [] \iff \text{transpose } ps = []$
using *transpose.elims* **by** blast

lemma *transpose-length*: $\text{length } ps = \text{length } (\text{transpose } ps)$
by (induction ps) auto

lemma *hd-transpose*: $ps \neq [] \implies \text{hd } (\text{transpose } ps) = \text{transpose-square } (\text{hd } ps)$
by (induction ps) (auto simp: transpose-square-def)

lemma *last-transpose*: $ps \neq [] \implies \text{last } (\text{transpose } ps) = \text{transpose-square } (\text{last } ps)$
proof (induction ps)
 case (Cons s_i ps)
 then show ?case
proof (cases ps = [])

```

    case True
    then show ?thesis using Cons by (auto simp: transpose-square-def)
next
    case False
    then show ?thesis using Cons transpose-nil by auto
qed
qed auto

```

```

lemma take-transpose:
  shows take k (transpose ps) = transpose (take k ps)
proof (induction ps arbitrary: k)
  case Nil
  then show ?case by auto
next
  case (Cons si ps)
  then obtain i j where si = (i,j) by force
  then have k = 0 ∨ k > 0 by auto
  then show ?case
  proof (elim disjE)
    assume k > 0
    then show ?thesis using Cons.IH by (auto simp: ⟨si = (i,j)⟩ take-Cons')
  qed auto
qed

```

```

lemma drop-transpose:
  shows drop k (transpose ps) = transpose (drop k ps)
proof (induction ps arbitrary: k)
  case Nil
  then show ?case by auto
next
  case (Cons si ps)
  then obtain i j where si = (i,j) by force
  then have k = 0 ∨ k > 0 by auto
  then show ?case
  proof (elim disjE)
    assume k > 0
    then show ?thesis using Cons.IH by (auto simp: ⟨si = (i,j)⟩ drop-Cons')
  qed auto
qed

```

```

lemma transpose-board-correct: si ∈ b ⟷ (transpose-square si) ∈ transpose-board b
  unfolding transpose-board-def transpose-square-def by (auto split: prod.splits)

```

```

lemma transpose-board: transpose-board (board n m) = board m n
  unfolding board-def using transpose-board-correct by (auto simp: transpose-square-def)

```

```

lemma insert-transpose-board:
  insert (transpose-square si) (transpose-board b) = transpose-board (insert si b)

```

```

unfolding transpose-board-def transpose-square-def by (auto split: prod.splits)

lemma transpose-board2: transpose-board (transpose-board b) = b
unfolding transpose-board-def by auto

lemma transpose-union: transpose-board (b1 ∪ b2) = transpose-board b1 ∪ trans-
pose-board b2
unfolding transpose-board-def by auto

lemma transpose-valid-step:
  valid-step si sj  $\longleftrightarrow$  valid-step (transpose-square si) (transpose-square sj)
unfolding valid-step-def transpose-square-def by (auto split: prod.splits)

lemma transpose-knights-path':
  assumes knights-path b ps
  shows knights-path (transpose-board b) (transpose ps)
  using assms
proof (induction rule: knights-path.induct)
  case (1 si)
  then have transpose-board {si} = {transpose-square si} transpose [si] = [transpose-square
si]
    using transpose-board-correct by (auto simp: transpose-square-def split: prod.splits)
  then show ?case by (auto intro: knights-path.intros)
next
  case (2 si b sj ps)
  then have prems: transpose-square si ∉ transpose-board b
    valid-step (transpose-square si) (transpose-square sj)
    and transpose (sj#ps) = transpose-square sj#transpose ps
    using 2 transpose-board-correct transpose-valid-step by auto
  then show ?case
    using 2 knights-path.intros(2)[OF prems] insert-transpose-board by auto
qed

corollary transpose-knights-path:
  assumes knights-path (board n m) ps
  shows knights-path (board m n) (transpose ps)
  using assms transpose-knights-path'[of board n m ps] by (auto simp: trans-
pose-board)

corollary transpose-knights-circuit:
  assumes knights-circuit (board n m) ps
  shows knights-circuit (board m n) (transpose ps)
  using assms
proof -
  have knights-path (board n m) ps and vs: valid-step (last ps) (hd ps)
    using assms unfolding knights-circuit-def by auto
  then have kp-t: knights-path (board m n) (transpose ps) and ps ≠ []
    using transpose-knights-path knights-path-non-nil by auto
  then have valid-step (last (transpose ps)) (hd (transpose ps))

```


using *vs hd-transpose last-transpose transpose-valid-step* by *auto*
 then show *?thesis using kp-t by (auto simp: knights-circuit-def)*
 qed

4.2 Mirror Paths and Boards

abbreviation $\text{min1 } ps \equiv \text{Min } ((\text{fst}) \text{ ' set } ps)$

abbreviation $\text{max1 } ps \equiv \text{Max } ((\text{fst}) \text{ ' set } ps)$

abbreviation $\text{min2 } ps \equiv \text{Min } ((\text{snd}) \text{ ' set } ps)$

abbreviation $\text{max2 } ps \equiv \text{Max } ((\text{snd}) \text{ ' set } ps)$

definition $\text{mirror1-square} :: \text{int} \Rightarrow \text{square} \Rightarrow \text{square}$ **where**
 $\text{mirror1-square } n \ s_i = (\text{case } s_i \text{ of } (i,j) \Rightarrow (n-i,j))$

fun $\text{mirror1-aux} :: \text{int} \Rightarrow \text{path} \Rightarrow \text{path}$ **where**
 $\text{mirror1-aux } n \ [] = []$
 $| \text{mirror1-aux } n \ (s_i \# ps) = (\text{mirror1-square } n \ s_i) \# \text{mirror1-aux } n \ ps$

definition $\text{mirror1 } ps = \text{mirror1-aux } (\text{max1 } ps + \text{min1 } ps) \ ps$

definition $\text{mirror1-board} :: \text{int} \Rightarrow \text{board} \Rightarrow \text{board}$ **where**
 $\text{mirror1-board } n \ b \equiv \{\text{mirror1-square } n \ s_i \mid s_i. s_i \in b\}$

definition $\text{mirror2-square} :: \text{int} \Rightarrow \text{square} \Rightarrow \text{square}$ **where**
 $\text{mirror2-square } m \ s_i = (\text{case } s_i \text{ of } (i,j) \Rightarrow (i,m-j))$

fun $\text{mirror2-aux} :: \text{int} \Rightarrow \text{path} \Rightarrow \text{path}$ **where**
 $\text{mirror2-aux } m \ [] = []$
 $| \text{mirror2-aux } m \ (s_i \# ps) = (\text{mirror2-square } m \ s_i) \# \text{mirror2-aux } m \ ps$

definition $\text{mirror2 } ps = \text{mirror2-aux } (\text{max2 } ps + \text{min2 } ps) \ ps$

definition $\text{mirror2-board} :: \text{int} \Rightarrow \text{board} \Rightarrow \text{board}$ **where**
 $\text{mirror2-board } m \ b \equiv \{\text{mirror2-square } m \ s_i \mid s_i. s_i \in b\}$

lemma $\text{mirror1-board-id} : \text{mirror1-board } (\text{int } n+1) \ (\text{board } n \ m) = \text{board } n \ m$ (**is -**
 $= ?b)$

proof

show $\text{mirror1-board } (\text{int } n+1) \ ?b \subseteq ?b$

proof

fix s_i'

assume $\text{assms} : s_i' \in \text{mirror1-board } (\text{int } n+1) \ ?b$

then obtain $i' \ j'$ **where** $[simp] : s_i' = (i',j')$ **by** *force*

then have $(i',j') \in \text{mirror1-board } (\text{int } n+1) \ ?b$

using assms **by** *auto*

then obtain $i \ j$ **where** $(i,j) \in ?b$ $\text{mirror1-square } (\text{int } n+1) \ (i,j) = (i',j')$

unfolding mirror1-board-def **by** *auto*

then have $1 \leq i \wedge i \leq \text{int } n \ 1 \leq j \wedge j \leq \text{int } m$ $i' = (\text{int } n+1) - i \ j' = j$

unfolding $\text{board-def mirror1-square-def}$ **by** *auto*

```

    then have  $1 \leq i' \wedge i' \leq \text{int } n$   $1 \leq j' \wedge j' \leq \text{int } m$ 
      by auto
    then show  $s_i' \in ?b$ 
      unfolding board-def by auto
  qed
next
show  $?b \subseteq \text{mirror1-board } (\text{int } n+1)$   $?b$ 
proof
  fix  $s_i$ 
  assume  $\text{assms}: s_i \in ?b$ 
  then obtain  $i\ j$  where  $[\text{simp}]: s_i = (i,j)$  by force
  then have  $(i,j) \in ?b$ 
    using  $\text{assms}$  by auto
  then have  $1 \leq i \wedge i \leq \text{int } n$   $1 \leq j \wedge j \leq \text{int } m$ 
    unfolding board-def by auto
  then obtain  $i'\ j'$  where  $i' = (\text{int } n+1) - i$   $j' = j$  by auto
  then have  $(i',j') \in ?b$   $\text{mirror1-square } (\text{int } n+1)$   $(i',j') = (i,j)$ 
    using  $\langle 1 \leq i \wedge i \leq \text{int } n \rangle$   $\langle 1 \leq j \wedge j \leq \text{int } m \rangle$ 
    unfolding mirror1-square-def by (auto simp: board-def)
  then show  $s_i \in \text{mirror1-board } (\text{int } n+1)$   $?b$ 
    unfolding mirror1-board-def by force
  qed
qed

lemma mirror2-board-id:  $\text{mirror2-board } (\text{int } m+1)$   $(\text{board } n\ m) = \text{board } n\ m$  (is -
=  $?b$ )
proof
  show  $\text{mirror2-board } (\text{int } m+1)$   $?b \subseteq ?b$ 
  proof
    fix  $s_i'$ 
    assume  $\text{assms}: s_i' \in \text{mirror2-board } (\text{int } m+1)$   $?b$ 
    then obtain  $i'\ j'$  where  $[\text{simp}]: s_i' = (i',j')$  by force
    then have  $(i',j') \in \text{mirror2-board } (\text{int } m+1)$   $?b$ 
      using  $\text{assms}$  by auto
    then obtain  $i\ j$  where  $(i,j) \in ?b$   $\text{mirror2-square } (\text{int } m+1)$   $(i,j) = (i',j')$ 
      unfolding mirror2-board-def by auto
    then have  $1 \leq i \wedge i \leq \text{int } n$   $1 \leq j \wedge j \leq \text{int } m$   $i' = i$   $j' = (\text{int } m+1) - j$ 
      unfolding board-def mirror2-square-def by auto
    then have  $1 \leq i' \wedge i' \leq \text{int } n$   $1 \leq j' \wedge j' \leq \text{int } m$ 
      by auto
    then show  $s_i' \in ?b$ 
      unfolding board-def by auto
    qed
  next
  show  $?b \subseteq \text{mirror2-board } (\text{int } m+1)$   $?b$ 
  proof
    fix  $s_i$ 
    assume  $\text{assms}: s_i \in ?b$ 
    then obtain  $i\ j$  where  $[\text{simp}]: s_i = (i,j)$  by force

```

```

then have  $(i,j) \in ?b$ 
  using assms by auto
then have  $1 \leq i \wedge i \leq \text{int } n \ 1 \leq j \wedge j \leq \text{int } m$ 
  unfolding board-def by auto
then obtain  $i' j'$  where  $i'=i \ j'=(\text{int } m+1)-j$  by auto
then have  $(i',j') \in ?b \ \text{mirror2-square } (\text{int } m+1) \ (i',j') = (i,j)$ 
  using  $\langle 1 \leq i \wedge i \leq \text{int } n \rangle \ \langle 1 \leq j \wedge j \leq \text{int } m \rangle$ 
  unfolding mirror2-square-def by (auto simp: board-def)
then show  $s_i \in \text{mirror2-board } (\text{int } m+1) \ ?b$ 
  unfolding mirror2-board-def by force
qed
qed

```

```

lemma knights-path-min1: knights-path (board  $n \ m$ )  $ps \implies \text{min1 } ps = 1$ 
proof -
  assume assms: knights-path (board  $n \ m$ )  $ps$ 
  then have  $\text{min } n \ m \geq 1$ 
    using knights-path-board-m-n-geq-1 by auto
  then have  $(1,1) \in \text{board } n \ m$  and  $ge-1: \forall (i,j) \in \text{board } n \ m. i \geq 1$ 
    unfolding board-def by auto
  then have finite: finite ((fst) ' board  $n \ m$ ) and
    non-empty: (fst) ' board  $n \ m \neq \{\}$  and
    mem-1:  $1 \in (\text{fst}) ' \text{board } n \ m$ 
    using board-finite by auto (metis fstI image-eqI)
  then have Min ((fst) ' board  $n \ m$ ) = 1
    using ge-1 by (auto simp: Min-eq-iff)
  then show ?thesis
    using assms knights-path-set-eq by auto
qed

```

```

lemma knights-path-min2: knights-path (board  $n \ m$ )  $ps \implies \text{min2 } ps = 1$ 
proof -
  assume assms: knights-path (board  $n \ m$ )  $ps$ 
  then have  $\text{min } n \ m \geq 1$ 
    using knights-path-board-m-n-geq-1 by auto
  then have  $(1,1) \in \text{board } n \ m$  and  $ge-1: \forall (i,j) \in \text{board } n \ m. j \geq 1$ 
    unfolding board-def by auto
  then have finite: finite ((snd) ' board  $n \ m$ ) and
    non-empty: (snd) ' board  $n \ m \neq \{\}$  and
    mem-1:  $1 \in (\text{snd}) ' \text{board } n \ m$ 
    using board-finite by auto (metis sndI image-eqI)
  then have Min ((snd) ' board  $n \ m$ ) = 1
    using ge-1 by (auto simp: Min-eq-iff)
  then show ?thesis
    using assms knights-path-set-eq by auto
qed

```

```

lemma knights-path-max1: knights-path (board  $n \ m$ )  $ps \implies \text{max1 } ps = \text{int } n$ 
proof -

```

```

assume assms: knights-path (board n m) ps
then have min n m  $\geq 1$ 
  using knights-path-board-m-n-geq-1 by auto
then have  $(\text{int } n, 1) \in \text{board } n \ m$  and leq-n:  $\forall (i,j) \in \text{board } n \ m. i \leq \text{int } n$ 
  unfolding board-def by auto
then have finite: finite ((fst) ‘ board n m) and
  non-empty: (fst) ‘ board n m  $\neq \{\}$  and
  mem-1:  $\text{int } n \in (\text{fst}) \text{ ‘ board } n \ m$ 
  using board-finite by auto (metis fstI image-eqI)
then have Max ((fst) ‘ board n m) = int n
  using leq-n by (auto simp: Max-eq-iff)
then show ?thesis
  using assms knights-path-set-eq by auto
qed

```

lemma *knights-path-max2*: *knights-path* (board *n m*) *ps* $\implies \text{max2 } ps = \text{int } m$
proof –

```

assume assms: knights-path (board n m) ps
then have min n m  $\geq 1$ 
  using knights-path-board-m-n-geq-1 by auto
then have  $(1, \text{int } m) \in \text{board } n \ m$  and leq-m:  $\forall (i,j) \in \text{board } n \ m. j \leq \text{int } m$ 
  unfolding board-def by auto
then have finite: finite ((snd) ‘ board n m) and
  non-empty: (snd) ‘ board n m  $\neq \{\}$  and
  mem-1:  $\text{int } m \in (\text{snd}) \text{ ‘ board } n \ m$ 
  using board-finite by auto (metis sndI image-eqI)
then have Max ((snd) ‘ board n m) = int m
  using leq-m by (auto simp: Max-eq-iff)
then show ?thesis
  using assms knights-path-set-eq by auto
qed

```

lemma *mirror1-aux-nil*: $ps = [] \longleftrightarrow \text{mirror1-aux } m \ ps = []$
using *mirror1-aux.elims* **by** *blast*

lemma *mirror1-nil*: $ps = [] \longleftrightarrow \text{mirror1 } ps = []$
unfolding *mirror1-def* **using** *mirror1-aux-nil* **by** *blast*

lemma *mirror2-aux-nil*: $ps = [] \longleftrightarrow \text{mirror2-aux } m \ ps = []$
using *mirror2-aux.elims* **by** *blast*

lemma *mirror2-nil*: $ps = [] \longleftrightarrow \text{mirror2 } ps = []$
unfolding *mirror2-def* **using** *mirror2-aux-nil* **by** *blast*

lemma *length-mirror1-aux*: $\text{length } ps = \text{length } (\text{mirror1-aux } n \ ps)$
by (*induction ps*) *auto*

lemma *length-mirror1*: $\text{length } ps = \text{length } (\text{mirror1 } ps)$
unfolding *mirror1-def* **using** *length-mirror1-aux* **by** *auto*

```

lemma length-mirror2-aux:  $\text{length } ps = \text{length } (\text{mirror2-aux } n \ ps)$ 
  by (induction ps) auto

lemma length-mirror2:  $\text{length } ps = \text{length } (\text{mirror2 } ps)$ 
  unfolding mirror2-def using length-mirror2-aux by auto

lemma mirror1-board-iff:  $s_i \notin b \iff \text{mirror1-square } n \ s_i \notin \text{mirror1-board } n \ b$ 
  unfolding mirror1-board-def mirror1-square-def by (auto split: prod.splits)

lemma mirror2-board-iff:  $s_i \notin b \iff \text{mirror2-square } n \ s_i \notin \text{mirror2-board } n \ b$ 
  unfolding mirror2-board-def mirror2-square-def by (auto split: prod.splits)

lemma insert-mirror1-board:
   $\text{insert } (\text{mirror1-square } n \ s_i) \ (\text{mirror1-board } n \ b) = \text{mirror1-board } n \ (\text{insert } s_i \ b)$ 
  unfolding mirror1-board-def mirror1-square-def by (auto split: prod.splits)

lemma insert-mirror2-board:
   $\text{insert } (\text{mirror2-square } n \ s_i) \ (\text{mirror2-board } n \ b) = \text{mirror2-board } n \ (\text{insert } s_i \ b)$ 
  unfolding mirror2-board-def mirror2-square-def by (auto split: prod.splits)

lemma  $(i::\text{int}) = i' + 1 \implies n - i = n - (i' + 1)$ 
  by auto

lemma valid-step-mirror1:
   $\text{valid-step } s_i \ s_j \iff \text{valid-step } (\text{mirror1-square } n \ s_i) \ (\text{mirror1-square } n \ s_j)$ 
proof
  assume assms: valid-step  $s_i \ s_j$ 
  obtain  $i \ j \ i' \ j'$  where [simp]:  $s_i = (i, j) \ s_j = (i', j')$  by force
  then have valid-step  $(n - i, j) \ (n - i', j')$ 
    using assms unfolding valid-step-def
    apply simp
    apply (elim disjE)
    apply auto
    done
  then show valid-step  $(\text{mirror1-square } n \ s_i) \ (\text{mirror1-square } n \ s_j)$ 
    unfolding mirror1-square-def by auto
next
  assume assms: valid-step  $(\text{mirror1-square } n \ s_i) \ (\text{mirror1-square } n \ s_j)$ 
  obtain  $i \ j \ i' \ j'$  where [simp]:  $s_i = (i, j) \ s_j = (i', j')$  by force
  then have valid-step  $(i, j) \ (i', j')$ 
    using assms unfolding valid-step-def mirror1-square-def
    apply simp
    apply (elim disjE)
    apply auto
    done
  then show valid-step  $s_i \ s_j$ 
    unfolding mirror1-square-def by auto
qed

```

lemma *valid-step-mirror2*:
 $valid_step\ s_i\ s_j \longleftrightarrow valid_step\ (mirror2_square\ m\ s_i)\ (mirror2_square\ m\ s_j)$

proof
assume *assms*: *valid-step* $s_i\ s_j$
obtain $i\ j\ i'\ j'$ **where** $[simp]: s_i = (i,j)\ s_j = (i',j')$ **by** *force*
then have *valid-step* $(i,m-j)\ (i',m-j')$
using *assms* **unfolding** *valid-step-def*
apply *simp*
apply $(elim\ disjE)$
apply *auto*
done
then show *valid-step* $(mirror2_square\ m\ s_i)\ (mirror2_square\ m\ s_j)$
unfolding *mirror2-square-def* **by** *auto*

next
assume *assms*: *valid-step* $(mirror2_square\ m\ s_i)\ (mirror2_square\ m\ s_j)$
obtain $i\ j\ i'\ j'$ **where** $[simp]: s_i = (i,j)\ s_j = (i',j')$ **by** *force*
then have *valid-step* $(i,j)\ (i',j')$
using *assms* **unfolding** *valid-step-def mirror2-square-def*
apply *simp*
apply $(elim\ disjE)$
apply *auto*
done
then show *valid-step* $s_i\ s_j$
unfolding *mirror1-square-def* **by** *auto*

qed

lemma *hd-mirror1*:
assumes *knights-path* $(board\ n\ m)\ ps\ hd\ ps = (i,j)$
shows $hd\ (mirror1\ ps) = (int\ n+1-i,j)$
using *assms*

proof –
have $hd\ (mirror1\ ps) = hd\ (mirror1_aux\ (int\ n+1)\ ps)$
unfolding *mirror1-def* **using** *assms knights-path-min1 knights-path-max1* **by** *auto*
also have $\dots = hd\ (mirror1_aux\ (int\ n+1)\ ((hd\ ps)\#(tl\ ps)))$
using *assms knights-path-non-nil* **by** $(metis\ list.collapse)$
also have $\dots = (int\ n+1-i,j)$
using *assms* **by** $(auto\ simp: mirror1-square-def)$
finally show *?thesis* .

qed

lemma *last-mirror1-aux*:
assumes $ps \neq []\ last\ ps = (i,j)$
shows $last\ (mirror1_aux\ n\ ps) = (n-i,j)$
using *assms*

proof *(induction ps)*
case $(Cons\ s_i\ ps)$
then show *?case*

using *mirror1-aux-nil Cons* **by** (*cases ps = []*) (*auto simp: mirror1-square-def*)
qed *auto*

lemma *last-mirror1*:

assumes *knight-path* (*board n m*) *ps last ps = (i,j)*
shows *last (mirror1 ps) = (int n+1-i,j)*
unfolding *mirror1-def* **using** *assms last-mirror1-aux knight-path-non-nil*
by (*simp add: knight-path-max1 knight-path-min1*)

lemma *hd-mirror2*:

assumes *knight-path* (*board n m*) *ps hd ps = (i,j)*
shows *hd (mirror2 ps) = (i,int m+1-j)*
using *assms*
proof –
 have *hd (mirror2 ps) = hd (mirror2-aux (int m+1) ps)*
 unfolding *mirror2-def* **using** *assms knight-path-min2 knight-path-max2* **by**
auto
 also have ... = *hd (mirror2-aux (int m+1) ((hd ps)#(tl ps)))*
 using *assms knight-path-non-nil* **by** (*metis list.collapse*)
 also have ... = (*i,int m+1-j*)
 using *assms* **by** (*auto simp: mirror2-square-def*)
 finally show ?thesis .
qed

lemma *last-mirror2-aux*:

assumes *ps ≠ [] last ps = (i,j)*
shows *last (mirror2-aux m ps) = (i,m-j)*
using *assms*
proof (*induction ps*)
 case (*Cons s_i ps*)
 then show ?case
 using *mirror2-aux-nil Cons* **by** (*cases ps = []*) (*auto simp: mirror2-square-def*)
qed *auto*

lemma *last-mirror2*:

assumes *knight-path* (*board n m*) *ps last ps = (i,j)*
shows *last (mirror2 ps) = (i,int m+1-j)*
unfolding *mirror2-def* **using** *assms last-mirror2-aux knight-path-non-nil*
by (*simp add: knight-path-max2 knight-path-min2*)

lemma *mirror1-aux-knight-path*:

assumes *knight-path b ps*
shows *knight-path (mirror1-board n b) (mirror1-aux n ps)*
using *assms*
proof (*induction rule: knight-path.induct*)
 case (*1 s_i*)
 then have *mirror1-board n {s_i} = {mirror1-square n s_i}*
 unfolding *mirror1-board-def* **by** *blast*
 then show ?case **by** (*auto intro: knight-path.intros*)

```

next
  case (2  $s_i$   $b$   $s_j$   $ps$ )
  then have prems: mirror1-square  $n$   $s_i$   $\notin$  mirror1-board  $n$   $b$ 
    valid-step (mirror1-square  $n$   $s_i$ ) (mirror1-square  $n$   $s_j$ )
    and mirror1-aux  $n$  ( $s_j \# ps$ ) = mirror1-square  $n$   $s_j \#$  mirror1-aux  $n$   $ps$ 
  using 2 mirror1-board-iff valid-step-mirror1 by auto
  then show ?case
  using 2 knights-path.intros(2)[OF prems] insert-mirror1-board by auto
qed

```

```

corollary mirror1-knights-path:
  assumes knights-path (board  $n$   $m$ )  $ps$ 
  shows knights-path (board  $n$   $m$ ) (mirror1  $ps$ )
  using assms
proof -
  have [simp]: min1  $ps$  = 1 max1  $ps$  = int  $n$ 
  using assms knights-path-min1 knights-path-max1 by auto
  then have mirror1-board (int  $n+1$ ) (board  $n$   $m$ ) = (board  $n$   $m$ )
  using mirror1-board-id by auto
  then have knights-path (board  $n$   $m$ ) (mirror1-aux (int  $n+1$ )  $ps$ )
  using assms mirror1-aux-knights-path[of board  $n$   $m$   $ps$  int  $n+1$ ] by auto
  then show ?thesis unfolding mirror1-def by auto
qed

```

```

lemma mirror2-aux-knights-path:
  assumes knights-path  $b$   $ps$ 
  shows knights-path (mirror2-board  $n$   $b$ ) (mirror2-aux  $n$   $ps$ )
  using assms
proof (induction rule: knights-path.induct)
  case (1  $s_i$ )
  then have mirror2-board  $n$   $\{s_i\}$  =  $\{mirror2-square$   $n$   $s_i\}$ 
  unfolding mirror2-board-def by blast
  then show ?case by (auto intro: knights-path.intros)
next
  case (2  $s_i$   $b$   $s_j$   $ps$ )
  then have prems: mirror2-square  $n$   $s_i$   $\notin$  mirror2-board  $n$   $b$ 
    valid-step (mirror2-square  $n$   $s_i$ ) (mirror2-square  $n$   $s_j$ )
    and mirror2-aux  $n$  ( $s_j \# ps$ ) = mirror2-square  $n$   $s_j \#$  mirror2-aux  $n$   $ps$ 
  using 2 mirror2-board-iff valid-step-mirror2 by auto
  then show ?case
  using 2 knights-path.intros(2)[OF prems] insert-mirror2-board by auto
qed

```

```

corollary mirror2-knights-path:
  assumes knights-path (board  $n$   $m$ )  $ps$ 
  shows knights-path (board  $n$   $m$ ) (mirror2  $ps$ )
proof -
  have [simp]: min2  $ps$  = 1 max2  $ps$  = int  $m$ 
  using assms knights-path-min2 knights-path-max2 by auto

```



```

then have mirror2-board (int m+1) (board n m) = (board n m)
using mirror2-board-id by auto
then have knights-path (board n m) (mirror2-aux (int m+1) ps)
using assms mirror2-aux-knights-path[of board n m ps int m+1] by auto
then show ?thesis unfolding mirror2-def by auto
qed

```

Transposing (*transpose*) and mirroring (along first axis *mirror1*) a Knight's path preserves the Knight's path's property. Tranpose+Mirror1 equals a 90deg-clockwise turn.

corollary *rot90-knights-path*:

```

assumes knights-path (board n m) ps
shows knights-path (board m n) (mirror1 (transpose ps))
using assms transpose-knights-path mirror1-knights-path by auto

```

lemma *hd-rot90-knights-path*:

```

assumes knights-path (board n m) ps hd ps = (i,j)
shows hd (mirror1 (transpose ps)) = (int m+1-j,i)
using assms

```

proof –

```

have hd (transpose ps) = (j,i) knights-path (board m n) (transpose ps)
using assms knights-path-non-nil hd-transpose transpose-knights-path
by (auto simp: transpose-square-def)
then show ?thesis using hd-mirror1 by auto

```

qed

lemma *last-rot90-knights-path*:

```

assumes knights-path (board n m) ps last ps = (i,j)
shows last (mirror1 (transpose ps)) = (int m+1-j,i)
using assms

```

proof –

```

have last (transpose ps) = (j,i) knights-path (board m n) (transpose ps)
using assms knights-path-non-nil last-transpose transpose-knights-path
by (auto simp: transpose-square-def)
then show ?thesis using last-mirror1 by auto

```

qed

5 Path and Board Translation

When constructing knight's paths for larger boards multiple knight's paths for smaller boards are concatenated. To concatenate paths the the coordinates in the path need to be translated. Therefore, simple auxiliary functions are provided.

5.1 Implementation of Path and Board Translation

Translate the coordinates for a path by (k_1, k_2) .

```

fun trans-path :: int × int ⇒ path ⇒ path where
  trans-path (k1,k2) [] = []
| trans-path (k1,k2) ((i,j)#xs) = (i+k1,j+k2)#(trans-path (k1,k2) xs)

```

Translate the coordinates of a board by (k_1, k_2) .

```

definition trans-board :: int × int ⇒ board ⇒ board where
  trans-board t b ≡ (case t of (k1,k2) ⇒ {(i+k1,j+k2) | i j. (i,j) ∈ b})

```

5.2 Correctness Proofs for Path and Board Translation

```

lemma trans-path-length: length ps = length (trans-path (k1,k2) ps)
by (induction ps) auto

```

```

lemma trans-path-non-nil: ps ≠ [] ⇒ trans-path (k1,k2) ps ≠ []
by (induction ps) auto

```

```

lemma trans-path-correct: (i,j) ∈ set ps ⟷ (i+k1,j+k2) ∈ set (trans-path (k1,k2) ps)

```

```

proof (induction ps)
  case (Cons si ps)
  then show ?case by (cases si) auto
qed auto

```

```

lemma trans-path-non-nil-last:
  ps ≠ [] ⇒ last (trans-path (k1,k2) ps) = last (trans-path (k1,k2) ((i,j)#ps))
using trans-path-non-nil by (induction ps) auto

```

```

lemma hd-trans-path:
  assumes ps ≠ [] hd ps = (i,j)
  shows hd (trans-path (k1,k2) ps) = (i+k1,j+k2)
using assms by (induction ps) auto

```

```

lemma last-trans-path:
  assumes ps ≠ [] last ps = (i,j)
  shows last (trans-path (k1,k2) ps) = (i+k1,j+k2)
using assms

```

```

proof (induction ps)
  case (Cons si ps)
  then show ?case
    using trans-path-non-nil-last[symmetric]
    apply (cases si)
    apply (cases ps = [])
    apply auto
  done
qed (auto)

```

```

lemma take-trans:
  shows take k (trans-path (k1,k2) ps) = trans-path (k1,k2) (take k ps)
proof (induction ps arbitrary: k)

```

```

    case Nil
    then show ?case by auto
next
    case (Cons si ps)
    then obtain i j where si = (i,j) by force
    then have k = 0 ∨ k > 0 by auto
    then show ?case
    proof (elim disjE)
      assume k > 0
      then show ?thesis using Cons.IH by (auto simp: ⟨si = (i,j)⟩ take-Cons')
    qed auto
qed

```

```

lemma drop-trans:
  shows drop k (trans-path (k1,k2) ps) = trans-path (k1,k2) (drop k ps)
proof (induction ps arbitrary: k)
  case Nil
  then show ?case by auto
next
  case (Cons si ps)
  then obtain i j where si = (i,j) by force
  then have k = 0 ∨ k > 0 by auto
  then show ?case
  proof (elim disjE)
    assume k > 0
    then show ?thesis using Cons.IH by (auto simp: ⟨si = (i,j)⟩ drop-Cons')
  qed auto
qed

```

```

lemma trans-board-correct: (i,j) ∈ b ⟷ (i+k1,j+k2) ∈ trans-board (k1,k2) b
  unfolding trans-board-def by auto

```

```

lemma board-subset: n1 ≤ n2 ⟹ m1 ≤ m2 ⟹ board n1 m1 ⊆ board n2 m2
  unfolding board-def by auto

```

Board concatenation

```

corollary board-concat:
  shows board n m1 ∪ trans-board (0,int m1) (board n m2) = board n (m1+m2)
  (is ?b1 ∪ ?b2 = ?b)
proof
  show ?b1 ∪ ?b2 ⊆ ?b unfolding board-def trans-board-def by auto
next
  show ?b ⊆ ?b1 ∪ ?b2
  proof
    fix x
    assume x ∈ ?b
    then obtain i j where x-split: x = (i,j) 1 ≤ i ∧ i ≤ int n 1 ≤ j ∧ j ≤ int
      (m1+m2)
    unfolding board-def by auto
  qed

```

```

then have  $j \leq \text{int } m_1 \vee (\text{int } m_1 < j \wedge j \leq \text{int } (m_1 + m_2))$  by auto
then show  $x \in ?b1 \cup ?b2$ 
proof
  assume  $j \leq \text{int } m_1$ 
  then show  $x \in ?b1 \cup ?b2$  using x-split unfolding board-def by auto
next
  assume asm:  $\text{int } m_1 < j \wedge j \leq \text{int } (m_1 + m_2)$ 
  then have  $(i, j - \text{int } m_1) \in \text{board } n \ m_2$  using x-split unfolding board-def by
auto
  then show  $x \in ?b1 \cup ?b2$ 
  using x-split asm trans-board-correct[of i j-int m1 board n m2 0 int m1] by
auto
  qed
qed
qed

```

lemma *transpose-trans-board*:

```

transpose-board (trans-board  $(k_1, k_2)$  b) = trans-board  $(k_2, k_1)$  (transpose-board b)
unfolding transpose-board-def trans-board-def by blast

```

corollary *board-concatT*:

```

shows board  $n_1 \ m \cup \text{trans-board } (0, \text{int } n_1) (\text{board } n_2 \ m) = \text{board } (n_1 + n_2) \ m$  (is
 $?b_1 \cup ?b_2 = ?b$ )

```

proof –

```

let  $?b_1 T = \text{board } m \ n_1$ 
let  $?b_2 T = \text{trans-board } (0, \text{int } n_1) (\text{board } m \ n_2)$ 
have  $?b_1 \cup ?b_2 = \text{transpose-board } (?b_1 T \cup ?b_2 T)$ 
using transpose-board2 transpose-union transpose-board transpose-trans-board
by auto
also have  $\dots = \text{transpose-board } (\text{board } m \ (n_1 + n_2))$ 
using board-concat by auto
also have  $\dots = \text{board } (n_1 + n_2) \ m$ 
using transpose-board by auto
finally show ?thesis .
qed

```

lemma *trans-valid-step*:

```

valid-step  $(i, j) \ (i', j') \implies \text{valid-step } (i + k_1, j + k_2) \ (i' + k_1, j' + k_2)$ 
unfolding valid-step-def by auto

```

Translating a path and a boards preserves the validity.

lemma *trans-knights-path*:

```

assumes knights-path b ps
shows knights-path (trans-board  $(k_1, k_2)$  b) (trans-path  $(k_1, k_2)$  ps)
using assms
proof (induction rule: knights-path.induct)
  case  $(2 \ s_i \ b \ s_j \ xs)$ 
  then obtain  $i \ j \ i' \ j'$  where split:  $s_i = (i, j) \ s_j = (i', j')$  by force
  let  $?s_i = (i + k_1, j + k_2)$ 

```

```

let ?sj=(i'+k1,j'+k2)
let ?xs=trans-path (k1,k2) xs
let ?b=trans-board (k1,k2) b
have simps: trans-path (k1,k2) (si#sj#xs) = ?si#?sj#?xs
      ?b ∪ {?si} = trans-board (k1,k2) (b ∪ {si})
  unfolding trans-board-def using split by auto
have ?si ∉ ?b valid-step ?si ?sj knights-path ?b (?sj#?xs)
  using 2 split trans-valid-step by (auto simp: trans-board-def)
then have knights-path (?b ∪ {?si}) (?si#?sj#?xs)
  using knights-path.intros by auto
then show ?case using simps by auto
qed (auto simp: trans-board-def intro: knights-path.intros)

```

Predicate that indicates if two squares s_i and s_j are adjacent in ps .

definition *step-in* :: *path* \Rightarrow *square* \Rightarrow *square* \Rightarrow *bool* **where**
step-in ps s_i $s_j \equiv (\exists k. 0 < k \wedge k < \text{length } ps \wedge \text{last } (\text{take } k \text{ } ps) = s_i \wedge \text{hd } (\text{drop } k \text{ } ps) = s_j)$

lemma *step-in-Cons*: *step-in* ps s_i $s_j \implies \text{step-in } (s_k \# ps) s_i s_j$
proof –
assume *step-in* ps s_i s_j
then obtain k **where** $0 < k \wedge k < \text{length } ps \wedge \text{last } (\text{take } k \text{ } ps) = s_i \wedge \text{hd } (\text{drop } k \text{ } ps) = s_j$
unfolding *step-in-def* **by** auto
then have $0 < k+1 \wedge k+1 < \text{length } (s_k \# ps)$
 $\text{last } (\text{take } (k+1) \text{ } (s_k \# ps)) = s_i \wedge \text{hd } (\text{drop } (k+1) \text{ } (s_k \# ps)) = s_j$
by auto
then show ?thesis
by (auto simp: *step-in-def*)
qed

lemma *step-in-append*: *step-in* ps s_i $s_j \implies \text{step-in } (ps @ ps') s_i s_j$
proof –
assume *step-in* ps s_i s_j
then obtain k **where** $0 < k \wedge k < \text{length } ps \wedge \text{last } (\text{take } k \text{ } ps) = s_i \wedge \text{hd } (\text{drop } k \text{ } ps) = s_j$
unfolding *step-in-def* **by** auto
then have $0 < k \wedge k < \text{length } (ps @ ps')$
 $\text{last } (\text{take } k \text{ } (ps @ ps')) = s_i \wedge \text{hd } (\text{drop } k \text{ } (ps @ ps')) = s_j$
by auto
then show ?thesis
by (auto simp: *step-in-def*)
qed

lemma *step-in-prepend*: *step-in* ps s_i $s_j \implies \text{step-in } (ps' @ ps) s_i s_j$
using *step-in-Cons* **by** (induction ps' arbitrary: ps) auto

lemma *step-in-valid-step*: *knights-path* b $ps \implies \text{step-in } ps s_i s_j \implies \text{valid-step } s_i s_j$

```

proof –
  assume assms: knights-path b ps step-in ps si sj
  then obtain k where k-prems:  $0 < k \wedge k < \text{length } ps$   $\text{last } (\text{take } k \text{ } ps) = s_i$   $\text{hd } (\text{drop } k \text{ } ps) = s_j$ 
    unfolding step-in-def by auto
  then have  $k = 1 \vee k > 1$  by auto
  then show ?thesis
  proof (elim disjE)
    assume  $k = 1$ 
    then obtain ps' where  $ps = s_i \# s_j \# ps'$ 
      using k-prems list-len-g-1-split by fastforce
    then show ?thesis
      using assms by (auto elim: knights-path.cases)
  next
    assume  $k > 1$ 
    then have  $0 < k-1 \wedge k-1 < \text{length } ps$ 
      using k-prems by auto
    then obtain b where knights-path b  $(\text{drop } (k-1) \text{ } ps)$ 
      using assms knights-path-split by blast

    obtain ps' where  $\text{drop } (k-1) \text{ } ps = s_i \# s_j \# ps'$ 
      using k-prems  $\langle 0 < k-1 \wedge k-1 < \text{length } ps \rangle$ 
    by (metis Cons-nth-drop-Suc Suc-diff-1 hd-drop-conv-nth last-snoc take-hd-drop)
    then show ?thesis
      using  $\langle \text{knights-path } b \text{ } (\text{drop } (k-1) \text{ } ps) \rangle$  by (auto elim: knights-path.cases)
  qed
qed

lemma trans-step-in:
   $\text{step-in } ps \text{ } (i,j) \text{ } (i',j') \implies \text{step-in } (\text{trans-path } (k_1,k_2) \text{ } ps) \text{ } (i+k_1,j+k_2) \text{ } (i'+k_1,j'+k_2)$ 
proof –
  let ?ps' = trans-path  $(k_1,k_2) \text{ } ps$ 
  assume step-in ps  $(i,j) \text{ } (i',j')$ 
  then obtain k where  $0 < k \wedge k < \text{length } ps$   $\text{last } (\text{take } k \text{ } ps) = (i,j)$   $\text{hd } (\text{drop } k \text{ } ps) = (i',j')$ 
    unfolding step-in-def by auto
  then have  $\text{take } k \text{ } ps \neq []$   $\text{drop } k \text{ } ps \neq []$  by fastforce
  then have  $0 < k \wedge k < \text{length } ?ps'$ 
     $\text{last } (\text{take } k \text{ } ?ps') = (i+k_1,j+k_2)$   $\text{hd } (\text{drop } k \text{ } ?ps') = (i'+k_1,j'+k_2)$ 
  using trans-path-length
     $\text{last-trans-path}[OF \text{ } \langle \text{take } k \text{ } ps \neq [] \rangle \text{ } \langle \text{last } (\text{take } k \text{ } ps) = (i,j) \rangle] \text{ take-trans}$ 
     $\text{hd-trans-path}[OF \text{ } \langle \text{drop } k \text{ } ps \neq [] \rangle \text{ } \langle \text{hd } (\text{drop } k \text{ } ps) = (i',j') \rangle] \text{ drop-trans}$ 
  by auto
  then show ?thesis
    by (auto simp: step-in-def)
  qed

```

```

lemma transpose-step-in:
   $\text{step-in } ps \text{ } s_i \text{ } s_j \implies \text{step-in } (\text{transpose } ps) \text{ } (\text{transpose-square } s_i) \text{ } (\text{transpose-square } s_j)$ 

```

$s_j)$
 (is - \implies step-in ?psT ?s_iT ?s_jT)
proof -
 assume step-in ps s_i s_j
 then obtain k where
 k-prems: $0 < k < \text{length } ps \text{ last } (take\ k\ ps) = s_i \text{ hd } (drop\ k\ ps) = s_j$
 unfolding step-in-def by auto
 then have non-nil: $take\ k\ ps \neq [] \text{ drop } k\ ps \neq []$ by fastforce+
 have $take\ k\ ?psT = \text{transpose } (take\ k\ ps) \text{ drop } k\ ?psT = \text{transpose } (drop\ k\ ps)$
 using take-transpose drop-transpose by auto
 then have $\text{last } (take\ k\ ?psT) = ?s_iT \text{ hd } (drop\ k\ ?psT) = ?s_jT$
 using non-nil k-prems hd-transpose last-transpose by auto
 then show step-in ?psT ?s_iT ?s_jT
 unfolding step-in-def using k-prems transpose-length by auto
 qed

lemma hd-take: $0 < k \implies \text{hd } xs = \text{hd } (take\ k\ xs)$
 by (induction xs) auto

lemma last-drop: $k < \text{length } xs \implies \text{last } xs = \text{last } (drop\ k\ xs)$
 by (induction xs) auto

Concatenate two knight's path on a $n \times m$ -board along the 2nd axis if the first path contains the step $s_i \rightarrow s_j$ and there are valid steps $s_i \rightarrow \text{hd } ps_2'$ and $s_j \rightarrow \text{last } ps_2'$, where ps_2' is ps_2 is translated by m_1 . An arbitrary step in ps_2 is preserved.

corollary knights-path-split-concat-si-prev:

assumes knights-path (board n m₁) ps₁ knights-path (board n m₂) ps₂
 step-in ps₁ s_i s_j hd ps₂ = (i_n,j_n) last ps₂ = (i_l,j_l) step-in ps₂ (i',j') (i',j')
 valid-step s_i (i_n,int m₁+j_n) valid-step (i_l,int m₁+j_l) s_j
 shows $\exists ps. \text{knights-path } (board\ n\ (m_1+m_2))\ ps \wedge \text{hd } ps = \text{hd } ps_1$
 $\wedge \text{last } ps = \text{last } ps_1 \wedge \text{step-in } ps\ (i, \text{int } m_1+j)\ (i', \text{int } m_1+j')$
 using assms

proof -
 let ?b₁=board n m₁
 let ?b₂=board n m₂
 let ?ps₂'=trans-path (0,int m₁) ps₂
 let ?b'=trans-board (0,int m₁) ?b₂
 have kp2': knights-path ?b' ?ps₂' using assms trans-knights-path by auto
 then have ?ps₂' $\neq []$ using knights-path-non-nil by auto

obtain k where k-prems:
 $0 < k < \text{length } ps_1 \text{ last } (take\ k\ ps_1) = s_i \text{ hd } (drop\ k\ ps_1) = s_j$
 using assms unfolding step-in-def by auto
 let ?ps=(take k ps₁) @ ?ps₂' @ (drop k ps₁)
 obtain b₁ b₂ where b-prems: knights-path b₁ (take k ps₁) knights-path b₂ (drop k ps₁)
 $b_1 \cup b_2 = ?b_1 \ b_1 \cap b_2 = \{\}$
 using assms $\langle 0 < k \rangle \langle k < \text{length } ps_1 \rangle$ knights-path-split by blast

have $hd\ ?ps_2' = (i_h, int\ m_1 + j_h)\ last\ ?ps_2' = (i_l, int\ m_1 + j_l)$
using *assms knights-path-non-nil hd-trans-path last-trans-path* **by** *auto*
then have $hd\ ?ps_2' = (i_h, int\ m_1 + j_h)\ last\ ((take\ k\ ps_1) @\ ?ps_2') = (i_l, int\ m_1 + j_l)$
using $\langle ?ps_2' \neq [] \rangle$ **by** *auto*
then have $vs: valid\ step\ (last\ (take\ k\ ps_1))\ (hd\ ?ps_2')$
 $valid\ step\ (last\ ((take\ k\ ps_1) @\ ?ps_2'))\ (hd\ (drop\ k\ ps_1))$
using *assms k-prems* **by** *auto*

have $?b_1 \cap ?b' = \{\}$ **unfolding** *board-def trans-board-def* **by** *auto*
then have $b_1 \cap ?b' = \{\} \wedge (b_1 \cup ?b') \cap b_2 = \{\}$ **using** *b-prems* **by** *blast*
then have *inter-empty*: $b_1 \cap ?b' = \{\} \wedge (b_1 \cup ?b') \cap b_2 = \{\}$ **by** *auto*

have *knights-path* $(b_1 \cup ?b')\ ((take\ k\ ps_1) @\ ?ps_2')$
using *kp2' b-prems inter-empty vs knights-path-append* **by** *auto*
then have *knights-path* $(b_1 \cup ?b' \cup b_2)\ ?ps$
using *b-prems inter-empty vs knights-path-append* [**where** $ps_1 = (take\ k\ ps_1) @\ ?ps_2'$] **by** *auto*
then have *knights-path* $(?b_1 \cup ?b')\ ?ps$
using *b-prems Un-commute Un-assoc* **by** *metis*
then have *kp*: *knights-path* $(board\ n\ (m_1 + m_2))\ ?ps$
using *board-concat* [*of* $n\ m_1\ m_2$] **by** *auto*

have $hd: hd\ ?ps = hd\ ps_1$
using *assms* $\langle 0 < k \rangle$ *knights-path-non-nil hd-take* **by** *auto*

have $last: last\ ?ps = last\ ps_1$
using *assms* $\langle k < length\ ps_1 \rangle$ *knights-path-non-nil last-drop* **by** *auto*

have *m-simps*: $j + int\ m_1 = int\ m_1 + j\ j' + int\ m_1 = int\ m_1 + j'$ **by** *auto*
have *si*: *step-in* $?ps\ (i, int\ m_1 + j)\ (i', int\ m_1 + j')$
using *assms step-in-append* [*OF step-in-prepend* [*OF trans-step-in*],
of $ps_2\ i\ j\ i'\ j'\ take\ k\ ps_1\ 0\ int\ m_1\ drop\ k\ ps_1$]
by (*auto simp: m-simps*)

show *thesis* **using** *kp hd last si* **by** *auto*
qed

lemma *len1-hd-last*: $length\ xs = 1 \implies hd\ xs = last\ xs$
by (*induction xs*) *auto*

Weaker version of $\llbracket knights-path\ (board\ ?n\ ?m_1)\ ?ps_1; knights-path\ (board\ ?n\ ?m_2)\ ?ps_2; step-in\ ?ps_1\ ?s_i\ ?s_j; hd\ ?ps_2 = (?i_h, ?j_h); last\ ?ps_2 = (?i_l, ?j_l); step-in\ ?ps_2\ (?i, ?j)\ (?i', ?j'); valid-step\ ?s_i\ (?i_h, int\ ?m_1 + ?j_h); valid-step\ (?i_l, int\ ?m_1 + ?j_l)\ ?s_j \rrbracket \implies \exists ps. knights-path\ (board\ ?n\ (?m_1 + ?m_2))\ ps \wedge hd\ ps = hd\ ?ps_1 \wedge last\ ps = last\ ?ps_1 \wedge step-in\ ps\ (?i, int\ ?m_1 + ?j)\ (?i', int\ ?m_1 + ?j')$.

corollary *knights-path-split-concat*:
assumes *knights-path* $(board\ n\ m_1)\ ps_1\ knights-path\ (board\ n\ m_2)\ ps_2$


```

      step-in ps1 si sj hd ps2 = (ih,jh) last ps2 = (il,jl)
      valid-step si (ih,int m1+jh) valid-step (il,int m1+jl) sj
    shows ∃ ps. knights-path (board n (m1+m2)) ps ∧ hd ps = hd ps1 ∧ last ps =
last ps1
  proof –
    have length ps2 = 1 ∨ length ps2 > 1
    using assms knights-path-non-nil by (meson length-0-conv less-one linorder-neqE-nat)
    then show ?thesis
    proof (elim disjE)
      let ?sk=(ih,int m1+jh)
      assume length ps2 = 1

      then have (ih,jh) = (il,jl)
      using assms len1-hd-last by metis
      then have valid-step si ?sk valid-step ?sk sj valid-step si sj
      using assms step-in-valid-step by auto
      then show ?thesis
      using valid-step-non-transitive by blast
    next
      assume length ps2 > 1
      then obtain i1 j1 i2 j2 ps2' where ps2 = (i1,j1)#(i2,j2)#ps2'
      using list-len-g-1-split by fastforce
      then have last (take 1 ps2) = (i1,j1) hd (drop 1 ps2) = (i2,j2) by auto
      then have step-in ps2 (i1,j1) (i2,j2) using ⟨length ps2 > 1⟩ by (auto simp:
step-in-def)
      then show ?thesis
      using assms knights-path-split-concat-si-prev by blast
    qed
  qed

```

Concatenate two knight's path on a $n \times m$ -board along the 1st axis.

corollary *knights-path-split-concatT*:

```

  assumes knights-path (board n1 m) ps1 knights-path (board n2 m) ps2
      step-in ps1 si sj hd ps2 = (ih,jh) last ps2 = (il,jl)
      valid-step si (int n1+ih,jh) valid-step (int n1+il,jl) sj
  shows ∃ ps. knights-path (board (n1+n2) m) ps ∧ hd ps = hd ps1 ∧ last ps =
last ps1
  using assms
  proof –
    let ?ps1T=transpose ps1
    let ?ps2T=transpose ps2
    have kps: knights-path (board m n1) ?ps1T knights-path (board m n2) ?ps2T
    using assms transpose-knights-path by auto

    let ?siT=transpose-square si
    let ?sjT=transpose-square sj
    have si: step-in ?ps1T ?siT ?sjT
    using assms transpose-step-in by auto
  
```

```

have  $ps_1 \neq []$   $ps_2 \neq []$ 
  using assms knights-path-non-nil by auto
then have  $hd\text{-}last2: hd\ ?ps_2 T = (j_h, i_h)$   $last\ ?ps_2 T = (j_l, i_l)$ 
  using assms hd-transpose last-transpose by (auto simp: transpose-square-def)

have  $vs: valid\text{-}step\ ?s_i T\ (j_h, int\ n_1 + i_h)$   $valid\text{-}step\ (j_l, int\ n_1 + i_l)\ ?s_j T$ 
  using assms transpose-valid-step by (auto simp: transpose-square-def split: prod.splits)

then obtain  $ps$  where
   $ps\text{-}prems: knights\text{-}path\ (board\ m\ (n_1 + n_2))\ ps\ hd\ ps = hd\ ?ps_1 T\ last\ ps = last\ ?ps_1 T$ 
  using knights-path-split-concat[OF kps si hd-last2 vs] by auto
then have  $ps \neq []$  using knights-path-non-nil by auto
let  $?psT = transpose\ ps$ 
have  $knights\text{-}path\ (board\ (n_1 + n_2)\ m)\ ?psT\ hd\ ?psT = hd\ ps_1\ last\ ?psT = last\ ps_1$ 
using  $\langle ps_1 \neq [] \rangle \langle ps \neq [] \rangle\ ps\text{-}prems\ transpose\text{-}knights\text{-}path\ hd\text{-}transpose\ last\text{-}transpose$ 
  by (auto simp: transpose2)
then show  $?thesis$  by auto
qed

```

Concatenate two Knight's path along the 2nd axis. There is a valid step from the last square in the first Knight's path ps_1 to the first square in the second Knight's path ps_2 .

corollary *knights-path-concat:*

```

assumes  $knights\text{-}path\ (board\ n\ m_1)\ ps_1\ knights\text{-}path\ (board\ n\ m_2)\ ps_2$ 
   $hd\ ps_2 = (i_h, j_h)\ valid\text{-}step\ (last\ ps_1)\ (i_h, int\ m_1 + j_h)$ 
shows  $knights\text{-}path\ (board\ n\ (m_1 + m_2))\ (ps_1 @ (trans\text{-}path\ (0, int\ m_1)\ ps_2))$ 
proof –
  let  $?ps_2' = trans\text{-}path\ (0, int\ m_1)\ ps_2$ 
  let  $?b = trans\text{-}board\ (0, int\ m_1)\ (board\ n\ m_2)$ 
  have  $inter\text{-}empty: board\ n\ m_1 \cap ?b = \{\}$ 
  unfolding board-def trans-board-def by auto
  have  $hd\ ?ps_2' = (i_h, int\ m_1 + j_h)$ 
  using assms knights-path-non-nil hd-trans-path by auto
  then have  $kp: knights\text{-}path\ (board\ n\ m_1)\ ps_1\ knights\text{-}path\ ?b\ ?ps_2'$  and
     $vs: valid\text{-}step\ (last\ ps_1)\ (hd\ ?ps_2')$ 
  using assms trans-knights-path by auto
  then show  $knights\text{-}path\ (board\ n\ (m_1 + m_2))\ (ps_1 @ ?ps_2')$ 
  using knights-path-append[OF kp inter-empty vs] board-concat by auto
qed

```

Concatenate two Knight's path along the 2nd axis. The first Knight's path end in $(2, m_1 - 1)$ (lower-right) and the second Knight's paths start in $(1, 1)$ (lower-left).

corollary *knights-path-lr-concat:*

```

assumes  $knights\text{-}path\ (board\ n\ m_1)\ ps_1\ knights\text{-}path\ (board\ n\ m_2)\ ps_2$ 

```

```

      last ps1 = (2,int m1-1) hd ps2 = (1,1)
    shows knights-path (board n (m1+m2)) (ps1 @ (trans-path (0,int m1) ps2))
  proof -
    have valid-step (last ps1) (1,int m1+1)
      using assms unfolding valid-step-def by auto
    then show ?thesis
      using assms knights-path-concat by auto
  qed

```

Concatenate two Knight's circuits along the 2nd axis. In the first Knight's path the squares $(2, m_1 - 1)$ and $(4, m_1)$ are adjacent and the second Knight's circuit starts in $(1, 1)$ (lower-left) and end in $(3, 2)$.

corollary *knights-circuit-lr-concat*:

```

    assumes knights-circuit (board n m1) ps1 knights-circuit (board n m2) ps2
      step-in ps1 (2,int m1-1) (4,int m1)
      hd ps2 = (1,1) last ps2 = (3,2) step-in ps2 (2,int m2-1) (4,int m2)
    shows ∃ ps. knights-circuit (board n (m1+m2)) ps ∧ step-in ps (2,int (m1+m2)-1)
      (4,int (m1+m2))
  proof -
    have kp1: knights-path (board n m1) ps1 and kp2: knights-path (board n m2) ps2

    and vs: valid-step (last ps1) (hd ps1)
    using assms unfolding knights-circuit-def by auto

    have m-simps: int m1 + (int m2-1) = int (m1+m2)-1 int m1 + int m2 = int
      (m1+m2) by auto

    have valid-step (2,int m1-1) (1,int m1+1) valid-step (3,int m1+2) (4,int m1)
      unfolding valid-step-def by auto
    then obtain ps where knights-path (board n (m1+m2)) ps hd ps = hd ps1 last
      ps = last ps1 and
      si: step-in ps (2,int (m1+m2)-1) (4,int (m1+m2))
    using assms kp1 kp2
      knights-path-split-concat-si-prev[of n m1 ps1 m2 ps2 (2,int m1-1)
        (4,int m1) 1 1 3 2 2 int m2-1 4 int m2]
      by (auto simp only: m-simps)
    then have knights-circuit (board n (m1+m2)) ps
      using vs by (auto simp: knights-circuit-def)
    then show ?thesis
      using si by auto
  qed

```

6 Path Parser and Path Construction

In this section functions are implemented to parse and construct paths. The parser converts the matrix representation $((nat\ list)\ list)$ used in [?] to a path $(path)$.

for debugging

```
fun test-path :: path  $\Rightarrow$  bool where
  test-path (si#sj#xs) = (step-checker si sj  $\wedge$  test-path (sj#xs))
| test-path - = True
```

```
fun f-opt :: ('a  $\Rightarrow$  'a)  $\Rightarrow$  'a option  $\Rightarrow$  'a option where
  f-opt - None = None
| f-opt f (Some a) = Some (f a)
```

```
fun add-opt-fst-sq :: int  $\Rightarrow$  square option  $\Rightarrow$  square option where
  add-opt-fst-sq - None = None
| add-opt-fst-sq k (Some (i,j)) = Some (k+i,j)
```

```
fun find-k-in-col :: nat  $\Rightarrow$  nat list  $\Rightarrow$  int option where
  find-k-in-col k [] = None
| find-k-in-col k (c#cs) = (if c = k then Some 1 else f-opt ((+) 1) (find-k-in-col k cs))
```

```
fun find-k-sqr :: nat  $\Rightarrow$  (nat list) list  $\Rightarrow$  square option where
  find-k-sqr k [] = None
| find-k-sqr k (r#rs) = (case find-k-in-col k r of
  None  $\Rightarrow$  f-opt ( $\lambda(i,j).$  (i+1,j)) (find-k-sqr k rs)
| Some j  $\Rightarrow$  Some (1,j))
```

Auxiliary function to easily parse pre-computed boards from paper.

```
fun to-sqrs :: nat  $\Rightarrow$  (nat list) list  $\Rightarrow$  path option where
  to-sqrs 0 rs = Some []
| to-sqrs k rs = (case find-k-sqr k rs of
  None  $\Rightarrow$  None
| Some si  $\Rightarrow$  f-opt ( $\lambda ps.$  ps@[si]) (to-sqrs (k-1) rs))
```

```
fun num-elems :: (nat list) list  $\Rightarrow$  nat where
  num-elems (r#rs) = length r * length (r#rs)
```

Convert a matrix (nat list list) to a path ((int \times int) list=path).

definition to-path rs \equiv to-sqrs (num-elems rs) (rev rs)

Example

```
value to-path
  [[3,22,13,16,5],
   [12,17,4,21,14],
   [23,2,15,6,9],
   [18,11,8,25,20],
   [1,24,19,10,7::nat]]
```

7 Paths for $5 \times m$ -Boards

Given here are knight's paths, $kp5xmlr$ and $kp5xmur$, for the $(5 \times m)$ -board that start in the lower-left corner for $m \in \{5, 6, 7, 8, 9\}$. The path $kp5xmlr$ ends in the lower-right corner, whereas the path $kp5xmur$ ends in the upper-right corner. The tables shows the visited squares numbered in ascending order.

abbreviation $b5x5 \equiv \text{board } 5 \ 5$

A Knight's path for the (5×5) -board that starts in the lower-left and ends in the lower-right.

3	22	13	16	5
12	17	4	21	14
23	2	15	6	9
18	11	8	25	20
1	24	19	10	7

abbreviation $kp5x5lr \equiv \text{the (to-path}$

$[[3, 22, 13, 16, 5],$
 $[12, 17, 4, 21, 14],$
 $[23, 2, 15, 6, 9],$
 $[18, 11, 8, 25, 20],$
 $[1, 24, 19, 10, 7]])$

lemma $kp\text{-}5x5\text{-}lr$: *knight's-path* $b5x5$ $kp5x5lr$

by (*simp only: knights-path-exec-simp*) *eval*

lemma $kp\text{-}5x5\text{-}lr\text{-}hd$: $hd \ kp5x5lr = (1, 1)$ **by** *eval*

lemma $kp\text{-}5x5\text{-}lr\text{-}last$: $last \ kp5x5lr = (2, 4)$ **by** *eval*

lemma $kp\text{-}5x5\text{-}lr\text{-}non\text{-}nil$: $kp5x5lr \neq []$ **by** *eval*

A Knight's path for the (5×5) -board that starts in the lower-left and ends in the upper-right.

7	12	15	20	5
16	21	6	25	14
11	8	13	4	19
22	17	2	9	24
1	10	23	18	3

abbreviation $kp5x5ur \equiv \text{the (to-path}$

$[[7, 12, 15, 20, 5],$
 $[16, 21, 6, 25, 14],$
 $[11, 8, 13, 4, 19],$

$[22,17,2,9,24],$
 $[1,10,23,18,3]]$)
lemma *kp-5x5-ur: knights-path b5x5 kp5x5ur*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-5x5-ur-hd: hd kp5x5ur = (1,1)* **by** *eval*

lemma *kp-5x5-ur-last: last kp5x5ur = (4,4)* **by** *eval*

lemma *kp-5x5-ur-non-nil: kp5x5ur ≠ []* **by** *eval*

abbreviation *b5x6 ≡ board 5 6*

A Knight's path for the (5×6) -board that starts in the lower-left and ends in the lower-right.

7	14	21	28	5	12
22	27	6	13	20	29
15	8	17	24	11	4
26	23	2	9	30	19
1	16	25	18	3	10

abbreviation *kp5x6lr ≡ the (to-path*
 $[[7,14,21,28,5,12],$
 $[22,27,6,13,20,29],$
 $[15,8,17,24,11,4],$
 $[26,23,2,9,30,19],$
 $[1,16,25,18,3,10]]$)
lemma *kp-5x6-lr: knights-path b5x6 kp5x6lr*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-5x6-lr-hd: hd kp5x6lr = (1,1)* **by** *eval*

lemma *kp-5x6-lr-last: last kp5x6lr = (2,5)* **by** *eval*

lemma *kp-5x6-lr-non-nil: kp5x6lr ≠ []* **by** *eval*

A Knight's path for the (5×6) -board that starts in the lower-left and ends in the upper-right.

3	10	29	20	5	12
28	19	4	11	30	21
9	2	17	24	13	6
18	27	8	15	22	25
1	16	23	26	7	14

abbreviation *kp5x6ur ≡ the (to-path*

$[[3,10,29,20,5,12],$
 $[28,19,4,11,30,21],$
 $[9,2,17,24,13,6],$
 $[18,27,8,15,22,25],$
 $[1,16,23,26,7,14]]$
lemma *kp-5x6-ur: knights-path b5x6 kp5x6ur*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-5x6-ur-hd: hd kp5x6ur = (1,1)* **by** *eval*

lemma *kp-5x6-ur-last: last kp5x6ur = (4,5)* **by** *eval*

lemma *kp-5x6-ur-non-nil: kp5x6ur ≠ []* **by** *eval*

abbreviation *b5x7 ≡ board 5 7*

A Knight's path for the (5×7) -board that starts in the lower-left and ends in the lower-right.

3	12	21	30	5	14	23
20	29	4	13	22	31	6
11	2	19	32	7	24	15
28	33	10	17	26	35	8
1	18	27	34	9	16	25

abbreviation *kp5x7lr ≡ the (to-path*
 $[[3,12,21,30,5,14,23],$
 $[20,29,4,13,22,31,6],$
 $[11,2,19,32,7,24,15],$
 $[28,33,10,17,26,35,8],$
 $[1,18,27,34,9,16,25]])$
lemma *kp-5x7-lr: knights-path b5x7 kp5x7lr*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-5x7-lr-hd: hd kp5x7lr = (1,1)* **by** *eval*

lemma *kp-5x7-lr-last: last kp5x7lr = (2,6)* **by** *eval*

lemma *kp-5x7-lr-non-nil: kp5x7lr ≠ []* **by** *eval*

A Knight's path for the (5×7) -board that starts in the lower-left and ends in the upper-right.

3	32	11	34	5	26	13
10	19	4	25	12	35	6
31	2	33	20	23	14	27
18	9	24	29	16	7	22
1	30	17	8	21	28	15

abbreviation $kp5x7ur \equiv$ the (to-path

$[[3,32,11,34,5,26,13],$
 $[10,19,4,25,12,35,6],$
 $[31,2,33,20,23,14,27],$
 $[18,9,24,29,16,7,22],$
 $[1,30,17,8,21,28,15]])$

lemma $kp-5x7-ur$: knights-path $b5x7$ $kp5x7ur$
by (simp only: knights-path-exec-simp) eval

lemma $kp-5x7-ur-hd$: $hd\ kp5x7ur = (1,1)$ **by** eval

lemma $kp-5x7-ur-last$: $last\ kp5x7ur = (4,6)$ **by** eval

lemma $kp-5x7-ur-non-nil$: $kp5x7ur \neq []$ **by** eval

abbreviation $b5x8 \equiv$ board 5 8

A Knight's path for the (5×8) -board that starts in the lower-left and ends in the lower-right.

3	12	37	26	5	14	17	28
34	23	4	13	36	27	6	15
11	2	35	38	25	16	29	18
22	33	24	9	20	31	40	7
1	10	21	32	39	8	19	30

abbreviation $kp5x8lr \equiv$ the (to-path

$[[3,12,37,26,5,14,17,28],$
 $[34,23,4,13,36,27,6,15],$
 $[11,2,35,38,25,16,29,18],$
 $[22,33,24,9,20,31,40,7],$
 $[1,10,21,32,39,8,19,30]])$

lemma $kp-5x8-lr$: knights-path $b5x8$ $kp5x8lr$
by (simp only: knights-path-exec-simp) eval

lemma $kp-5x8-lr-hd$: $hd\ kp5x8lr = (1,1)$ **by** eval

lemma $kp-5x8-lr-last$: $last\ kp5x8lr = (2,7)$ **by** eval

lemma $kp-5x8-lr-non-nil$: $kp5x8lr \neq []$ **by** eval

A Knight's path for the (5×8) -board that starts in the lower-left and ends in the upper-right.

33	8	17	38	35	6	15	24
18	37	34	7	16	25	40	5
9	32	29	36	39	14	23	26
30	19	2	11	28	21	4	13
1	10	31	20	3	12	27	22

abbreviation $kp5x8ur \equiv$ the (to-path

$[[33,8,17,38,35,6,15,24],$
 $[18,37,34,7,16,25,40,5],$
 $[9,32,29,36,39,14,23,26],$
 $[30,19,2,11,28,21,4,13],$
 $[1,10,31,20,3,12,27,22]]$)

lemma $kp-5x8-ur$: *knight's-path* $b5x8$ $kp5x8ur$
by (*simp only: knight's-path-exec-simp*) *eval*

lemma $kp-5x8-ur-hd$: $hd\ kp5x8ur = (1,1)$ **by** *eval*

lemma $kp-5x8-ur-last$: $last\ kp5x8ur = (4,7)$ **by** *eval*

lemma $kp-5x8-ur-non-nil$: $kp5x8ur \neq []$ **by** *eval*

abbreviation $b5x9 \equiv$ board 5 9

A Knight's path for the (5×9) -board that starts in the lower-left and ends in the lower-right.

9	4	11	16	23	42	33	36	25
12	17	8	3	32	37	24	41	34
5	10	15	20	43	22	35	26	29
18	13	2	7	38	31	28	45	40
1	6	19	14	21	44	39	30	27

abbreviation $kp5x9lr \equiv$ the (to-path

$[9,4,11,16,23,42,33,36,25],$
 $[12,17,8,3,32,37,24,41,34],$
 $[5,10,15,20,43,22,35,26,29],$
 $[18,13,2,7,38,31,28,45,40],$
 $[1,6,19,14,21,44,39,30,27]]$)

lemma $kp-5x9-lr$: *knight's-path* $b5x9$ $kp5x9lr$
by (*simp only: knight's-path-exec-simp*) *eval*

lemma $kp-5x9-lr-hd$: $hd\ kp5x9lr = (1,1)$ **by** *eval*

lemma $kp-5x9-lr-last$: $last\ kp5x9lr = (2,8)$ **by** *eval*

lemma $kp-5x9-lr-non-nil$: $kp5x9lr \neq []$ **by** *eval*

A Knight's path for the (5×9) -board that starts in the lower-left and ends in the upper-right.

9	4	11	16	27	32	35	40	25
12	17	8	3	36	41	26	45	34
5	10	15	20	31	28	33	24	39
18	13	2	7	42	37	22	29	44
1	6	19	14	21	30	43	38	23

abbreviation $kp5x9ur \equiv$ the (to-path

$[[9,4,11,16,27,32,35,40,25],$
 $[12,17,8,3,36,41,26,45,34],$
 $[5,10,15,20,31,28,33,24,39],$
 $[18,13,2,7,42,37,22,29,44],$
 $[1,6,19,14,21,30,43,38,23]]$)

lemma $kp-5x9-ur$: *knight's-path* $b5x9$ $kp5x9ur$
by (*simp only: knights-path-exec-simp*) *eval*

lemma $kp-5x9-ur-hd$: $hd\ kp5x9ur = (1,1)$ **by** *eval*

lemma $kp-5x9-ur-last$: $last\ kp5x9ur = (4,8)$ **by** *eval*

lemma $kp-5x9-ur-non-nil$: $kp5x9ur \neq []$ **by** *eval*

lemmas $kp-5xm-lr =$

$kp-5x5-lr\ kp-5x5-lr-hd\ kp-5x5-lr-last\ kp-5x5-lr-non-nil$
 $kp-5x6-lr\ kp-5x6-lr-hd\ kp-5x6-lr-last\ kp-5x6-lr-non-nil$
 $kp-5x7-lr\ kp-5x7-lr-hd\ kp-5x7-lr-last\ kp-5x7-lr-non-nil$
 $kp-5x8-lr\ kp-5x8-lr-hd\ kp-5x8-lr-last\ kp-5x8-lr-non-nil$
 $kp-5x9-lr\ kp-5x9-lr-hd\ kp-5x9-lr-last\ kp-5x9-lr-non-nil$

lemmas $kp-5xm-ur =$

$kp-5x5-ur\ kp-5x5-ur-hd\ kp-5x5-ur-last\ kp-5x5-ur-non-nil$
 $kp-5x6-ur\ kp-5x6-ur-hd\ kp-5x6-ur-last\ kp-5x6-ur-non-nil$
 $kp-5x7-ur\ kp-5x7-ur-hd\ kp-5x7-ur-last\ kp-5x7-ur-non-nil$
 $kp-5x8-ur\ kp-5x8-ur-hd\ kp-5x8-ur-last\ kp-5x8-ur-non-nil$
 $kp-5x9-ur\ kp-5x9-ur-hd\ kp-5x9-ur-last\ kp-5x9-ur-non-nil$

For every $5 \times m$ -board with $m \geq 5$ there exists a knight's path that starts in $(1,1)$ (bottom-left) and ends in $(2,m-1)$ (bottom-right).

lemma *knight's-path-5xm-lr-exists*:

assumes $m \geq 5$

shows $\exists ps.$ *knight's-path* (board 5 m) $ps \wedge hd\ ps = (1,1) \wedge last\ ps = (2,int\ m-1)$

using *assms*

proof (*induction m rule: less-induct*)

case (*less m*)

then have $m \in \{5,6,7,8,9\} \vee 5 \leq m-5$ **by** *auto*

then show *?case*

```

proof (elim disjE)
  assume  $m \in \{5,6,7,8,9\}$ 
  then show ?thesis using kp-5xm-lr by fastforce
next
  assume  $m \geq 5$ 
  then obtain  $ps_1$  where  $ps_1$ -IH: knights-path (board 5 (m-5))  $ps_1$  hd  $ps_1 =$ 
(1,1)
      last  $ps_1 = (2, \text{int } (m-5)-1)$   $ps_1 \neq []$ 
  using less.IH[of m-5] knights-path-non-nil by auto

  let  $?ps_2 = kp5x5lr$ 
  let  $?ps_2' = ps_1 @ \text{trans-path } (0, \text{int } (m-5)) ?ps_2$ 
  have knights-path b5x5  $?ps_2$  hd  $?ps_2 = (1, 1)$   $?ps_2 \neq []$  last  $?ps_2 = (2,4)$ 
    using kp-5xm-lr by auto
  then have 1: knights-path (board 5 m)  $?ps_2'$ 
    using m-ge  $ps_1$ -IH knights-path-lr-concat[of 5 m-5  $ps_1$  5  $?ps_2$ ] by auto

  have 2: hd  $?ps_2' = (1,1)$  using  $ps_1$ -IH by auto

  have last (trans-path (0, int (m-5))  $?ps_2$ ) = (2, int m-1)
    using m-ge last-trans-path[OF  $\langle ?ps_2 \neq [] \rangle \langle \text{last } ?ps_2 = (2,4) \rangle$ ] by auto
  then have 3: last  $?ps_2' = (2, \text{int } m-1)$ 
    using last-appendR[OF trans-path-non-nil[OF  $\langle ?ps_2 \neq [] \rangle$ , symmetric] by
metis

  show ?thesis using 1 2 3 by auto
qed
qed

```

For every $5 \times m$ -board with $m \geq 5$ there exists a knight's path that starts in (1,1) (bottom-left) and ends in (4, m-1) (top-right).

lemma knights-path-5xm-ur-exists:

```

assumes  $m \geq 5$ 
shows  $\exists ps. \text{knights-path (board 5 m) } ps \wedge \text{hd } ps = (1,1) \wedge \text{last } ps = (4, \text{int } m-1)$ 
using assms
proof -
  have  $m \in \{5,6,7,8,9\} \vee 5 \leq m-5$  using assms by auto
  then show ?thesis
  proof (elim disjE)
    assume  $m \in \{5,6,7,8,9\}$ 
    then show ?thesis using kp-5xm-ur by fastforce
  next
    assume  $m \geq 5$ 
    then obtain  $ps_1$  where  $ps$ -prems: knights-path (board 5 (m-5))  $ps_1$  hd  $ps_1 =$ 
(1,1)
      last  $ps_1 = (2, \text{int } (m-5)-1)$   $ps_1 \neq []$ 
    using knights-path-5xm-lr-exists[of (m-5)] knights-path-non-nil by auto
  let  $?ps_2 = kp5x5ur$ 
  let  $?ps' = ps_1 @ \text{trans-path } (0, \text{int } (m-5)) ?ps_2$ 

```

```

have knights-path b5x5 ?ps2 hd ?ps2 = (1, 1) ?ps2 ≠ []
  last ?ps2 = (4,4)
  using kp-5xm-ur by auto
then have 1: knights-path (board 5 m) ?ps'
  using m-ge ps-prems knights-path-lr-concat[of 5 m-5 ps1 5 ?ps2] by auto

have 2: hd ?ps' = (1,1) using ps-prems by auto

have last (trans-path (0,int (m-5)) ?ps2) = (4,int m-1)
  using m-ge last-trans-path[OF ‹?ps2 ≠ []› ‹last ?ps2 = (4,4)›] by auto
then have 3: last ?ps' = (4,int m-1)
  using last-appendR[OF trans-path-non-nil[OF ‹?ps2 ≠ []›],symmetric] by
metis

```

```

show ?thesis using 1 2 3 by auto
qed
qed

```

$5 \leq ?m \implies \exists ps. \text{knights-path (board 5 ?m) } ps \wedge \text{hd } ps = (1, 1) \wedge \text{last } ps = (2, \text{int } ?m - 1)$ and $5 \leq ?m \implies \exists ps. \text{knights-path (board 5 ?m) } ps \wedge \text{hd } ps = (1, 1) \wedge \text{last } ps = (2, \text{int } ?m - 1)$ formalize Lemma 1 from [?].

lemmas *knights-path-5xm-exists = knights-path-5xm-lr-exists knights-path-5xm-ur-exists*

8 Paths and Circuits for $6 \times m$ -Boards

abbreviation *b6x5* \equiv *board 6 5*

A Knight's path for the (6×5) -board that starts in the lower-left and ends in the upper-left.

10	19	4	29	12
3	30	11	20	5
18	9	24	13	28
25	2	17	6	21
16	23	8	27	14
1	26	15	22	7

abbreviation *kp6x5ul* \equiv *the (to-path*

```

[[10,19,4,29,12],
 [3,30,11,20,5],
 [18,9,24,13,28],
 [25,2,17,6,21],
 [16,23,8,27,14],
 [1,26,15,22,7]]

```

lemma *kp-6x5-ul: knights-path b6x5 kp6x5ul*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-6x5-ul-hd*: $hd\ kp6x5ul = (1,1)$ **by** *eval*

lemma *kp-6x5-ul-last*: $last\ kp6x5ul = (5,2)$ **by** *eval*

lemma *kp-6x5-ul-non-nil*: $kp6x5ul \neq []$ **by** *eval*

A Knight's circuit for the (6×5) -board.

16	9	6	27	18
7	26	17	14	5
10	15	8	19	28
25	30	23	4	13
22	11	2	29	20
1	24	21	12	3

abbreviation *kc6x5* \equiv *the (to-path*

$[[16,9,6,27,18],$
 $[7,26,17,14,5],$
 $[10,15,8,19,28],$
 $[25,30,23,4,13],$
 $[22,11,2,29,20],$
 $[1,24,21,12,3]]]$

lemma *kc-6x5*: *knight-circuit b6x5 kc6x5*
by (*simp only: knight-circuit-exec-simp*) *eval*

lemma *kc-6x5-hd*: $hd\ kc6x5 = (1,1)$ **by** *eval*

lemma *kc-6x5-non-nil*: $kc6x5 \neq []$ **by** *eval*

abbreviation *b6x6* \equiv *board 6 6*

The path given for the 6×6 -board that ends in the upper-left is wrong. The Knight cannot move from square 26 to square 27.

14	23	6	28	12	21
7	36	13	22	5	27
24	15	29	35	20	11
30	8	17	26	34	4
16	25	2	32	10	19
1	31	9	18	3	33

abbreviation *kp6x6ul-wrong* \equiv *the (to-path*

$[[14,23,6,28,12,21],$
 $[7,36,13,22,5,27],$
 $[24,15,29,35,20,11],$
 $[30,8,17,26,34,4],$
 $[16,25,2,32,10,19],$

$[1,31,9,18,3,33]]$)

value *path-checker* (*board-exec* 6 6) *kp6x6ul-wrong*

I have computed a correct Knight's path for the 6×6 -board that ends in the upper-left. A Knight's path for the (6×6) -board that starts in the lower-left and ends in the upper-left.

8	25	10	21	6	23
11	36	7	24	33	20
26	9	34	3	22	5
35	12	15	30	19	32
14	27	2	17	4	29
1	16	13	28	31	18

abbreviation *kp6x6ul* \equiv *the (to-path*

$[[8,25,10,21,6,23],$
 $[11,36,7,24,33,20],$
 $[26,9,34,3,22,5],$
 $[35,12,15,30,19,32],$
 $[14,27,2,17,4,29],$
 $[1,16,13,28,31,18]]$)

lemma *kp-6x6-ul: knights-path b6x6 kp6x6ul*

by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-6x6-ul-hd: hd kp6x6ul = (1,1)* **by** *eval*

lemma *kp-6x6-ul-last: last kp6x6ul = (5,2)* **by** *eval*

lemma *kp-6x6-ul-non-nil: kp6x6ul $\neq []$* **by** *eval*

A Knight's circuit for the (6×6) -board.

4	25	34	15	18	7
35	14	5	8	33	16
24	3	26	17	6	19
13	36	23	30	9	32
22	27	2	11	20	29
1	12	21	28	31	10

abbreviation *kc6x6* \equiv *the (to-path*

$[[4,25,34,15,18,7],$
 $[35,14,5,8,33,16],$
 $[24,3,26,17,6,19],$
 $[13,36,23,30,9,32],$
 $[22,27,2,11,20,29],$
 $[1,12,21,28,31,10]]$)

lemma *kc-6x6: knights-circuit b6x6 kc6x6*
by (*simp only: knights-circuit-exec-simp*) *eval*

lemma *kc-6x6-hd: hd kc6x6 = (1,1)* **by** *eval*

lemma *kc-6x6-non-nil: kc6x6 ≠ []* **by** *eval*

abbreviation *b6x7 ≡ board 6 7*

A Knight's path for the (6×7) -board that starts in the lower-left and ends in the upper-left.

18	23	8	39	16	25	6
9	42	17	24	7	40	15
22	19	32	41	38	5	26
33	10	21	28	31	14	37
20	29	2	35	12	27	4
1	34	11	30	3	36	13

abbreviation *kp6x7ul ≡ the (to-path*

[[18,23,8,39,16,25,6],
[9,42,17,24,7,40,15],
[22,19,32,41,38,5,26],
[33,10,21,28,31,14,37],
[20,29,2,35,12,27,4],
[1,34,11,30,3,36,13]])

lemma *kp-6x7-ul: knights-path b6x7 kp6x7ul*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-6x7-ul-hd: hd kp6x7ul = (1,1)* **by** *eval*

lemma *kp-6x7-ul-last: last kp6x7ul = (5,2)* **by** *eval*

lemma *kp-6x7-ul-non-nil: kp6x7ul ≠ []* **by** *eval*

A Knight's circuit for the (6×7) -board.

26	37	8	17	28	31	6
9	18	27	36	7	16	29
38	25	10	19	30	5	32
11	42	23	40	35	20	15
24	39	2	13	22	33	4
1	12	41	34	3	14	21

abbreviation *kc6x7 ≡ the (to-path*

[[26,37,8,17,28,31,6],
[9,18,27,36,7,16,29],

$[38,25,10,19,30,5,32],$
 $[11,42,23,40,35,20,15],$
 $[24,39,2,13,22,33,4],$
 $[1,12,41,34,3,14,21]]$)
lemma *kc-6x7: knights-circuit b6x7 kc6x7*
by (*simp only: knights-circuit-exec-simp*) *eval*

lemma *kc-6x7-hd: hd kc6x7 = (1,1)* **by** *eval*

lemma *kc-6x7-non-nil: kc6x7 ≠ []* **by** *eval*

abbreviation *b6x8 ≡ board 6 8*

A Knight's path for the (6×8) -board that starts in the lower-left and ends in the upper-left.

18	31	8	35	16	33	6	45
9	48	17	32	7	46	15	26
30	19	36	47	34	27	44	5
37	10	21	28	43	40	25	14
20	29	2	39	12	23	4	41
1	38	11	22	3	42	13	24

abbreviation *kp6x8ul ≡ the (to-path*
 $[[18,31,8,35,16,33,6,45],$
 $[9,48,17,32,7,46,15,26],$
 $[30,19,36,47,34,27,44,5],$
 $[37,10,21,28,43,40,25,14],$
 $[20,29,2,39,12,23,4,41],$
 $[1,38,11,22,3,42,13,24]]$)
lemma *kp-6x8-ul: knights-path b6x8 kp6x8ul*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-6x8-ul-hd: hd kp6x8ul = (1,1)* **by** *eval*

lemma *kp-6x8-ul-last: last kp6x8ul = (5,2)* **by** *eval*

lemma *kp-6x8-ul-non-nil: kp6x8ul ≠ []* **by** *eval*

A Knight's circuit for the (6×8) -board.

30	35	8	15	28	39	6	13
9	16	29	36	7	14	27	38
34	31	10	23	40	37	12	5
17	48	33	46	11	22	41	26
32	45	2	19	24	43	4	21
1	18	47	44	3	20	25	42

abbreviation $kc6x8 \equiv$ the (to-path

$[[30,35,8,15,28,39,6,13],$
 $[9,16,29,36,7,14,27,38],$
 $[34,31,10,23,40,37,12,5],$
 $[17,48,33,46,11,22,41,26],$
 $[32,45,2,19,24,43,4,21],$
 $[1,18,47,44,3,20,25,42]])$

lemma $kc-6x8$: knights-circuit $b6x8$ $kc6x8$

by (simp only: knights-circuit-exec-simp) eval

lemma $kc-6x8-hd$: $hd\ kc6x8 = (1,1)$ **by** eval

lemma $kc-6x8-non-nil$: $kc6x8 \neq []$ **by** eval

abbreviation $b6x9 \equiv$ board 6 9

A Knight's path for the (6×9) -board that starts in the lower-left and ends in the upper-left.

22	45	10	53	20	47	8	35	18
11	54	21	46	9	36	19	48	7
44	23	42	37	52	49	32	17	34
41	12	25	50	27	38	29	6	31
24	43	2	39	14	51	4	33	16
1	40	13	26	3	28	15	30	5

abbreviation $kp6x9ul \equiv$ the (to-path

$[[22,45,10,53,20,47,8,35,18],$
 $[11,54,21,46,9,36,19,48,7],$
 $[44,23,42,37,52,49,32,17,34],$
 $[41,12,25,50,27,38,29,6,31],$
 $[24,43,2,39,14,51,4,33,16],$
 $[1,40,13,26,3,28,15,30,5]])$

lemma $kp-6x9-ul$: knights-path $b6x9$ $kp6x9ul$

by (simp only: knights-path-exec-simp) eval

lemma $kp-6x9-ul-hd$: $hd\ kp6x9ul = (1,1)$ **by** eval

lemma $kp-6x9-ul-last$: $last\ kp6x9ul = (5,2)$ **by** eval

lemma $kp-6x9-ul-non-nil$: $kp6x9ul \neq []$ **by** eval

A Knight's circuit for the (6×9) -board.

14	49	4	51	24	39	6	29	22
3	52	13	40	5	32	23	42	7
48	15	50	25	38	41	28	21	30
53	2	37	12	33	26	31	8	43
16	47	54	35	18	45	10	27	20
1	36	17	46	11	34	19	44	9

abbreviation $kc6x9 \equiv$ the (to-path

[[14,49,4,51,24,39,6,29,22],
 [3,52,13,40,5,32,23,42,7],
 [48,15,50,25,38,41,28,21,30],
 [53,2,37,12,33,26,31,8,43],
 [16,47,54,35,18,45,10,27,20],
 [1,36,17,46,11,34,19,44,9]])

lemma $kc-6x9$: knights-circuit $b6x9$ $kc6x9$
 by (simp only: knights-circuit-exec-simp) eval

lemma $kc-6x9-hd$: $hd\ kc6x9 = (1,1)$ by eval

lemma $kc-6x9-non-nil$: $kc6x9 \neq []$ by eval

lemmas $kp-6xm-ul =$

$kp-6x5-ul\ kp-6x5-ul-hd\ kp-6x5-ul-last\ kp-6x5-ul-non-nil$
 $kp-6x6-ul\ kp-6x6-ul-hd\ kp-6x6-ul-last\ kp-6x6-ul-non-nil$
 $kp-6x7-ul\ kp-6x7-ul-hd\ kp-6x7-ul-last\ kp-6x7-ul-non-nil$
 $kp-6x8-ul\ kp-6x8-ul-hd\ kp-6x8-ul-last\ kp-6x8-ul-non-nil$
 $kp-6x9-ul\ kp-6x9-ul-hd\ kp-6x9-ul-last\ kp-6x9-ul-non-nil$

lemmas $kc-6xm =$

$kc-6x5\ kc-6x5-hd\ kc-6x5-non-nil$
 $kc-6x6\ kc-6x6-hd\ kc-6x6-non-nil$
 $kc-6x7\ kc-6x7-hd\ kc-6x7-non-nil$
 $kc-6x8\ kc-6x8-hd\ kc-6x8-non-nil$
 $kc-6x9\ kc-6x9-hd\ kc-6x9-non-nil$

For every $6 \times m$ -board with $m \geq 5$ there exists a knight's path that starts in $(1,1)$ (bottom-left) and ends in $(5,2)$ (top-left).

lemma $knights-path-6xm-ul-exists$:

assumes $m \geq 5$

shows $\exists ps. knights-path\ (board\ 6\ m)\ ps \wedge hd\ ps = (1,1) \wedge last\ ps = (5,2)$

using $assms$

proof (induction m rule: less-induct)

case (less m)

then have $m \in \{5,6,7,8,9\} \vee 5 \leq m-5$ **by** auto

then show $?case$

proof (elim $disjE$)

assume $m \in \{5,6,7,8,9\}$

then show $?thesis$ **using** $kp-6xm-ul$ **by** fastforce

```

next
  let ?ps1=kp6x5ul
  let ?b1=board 6 5
  have ps1-prems: knights-path ?b1 ?ps1 hd ?ps1 = (1,1) last ?ps1 = (5,2)
    using kp-6xm-ul by auto
  assume m-ge: 5 ≤ m-5
  then obtain ps2 where ps2-IH: knights-path (board 6 (m-5)) ps2 hd ps2 =
    (1,1)
    last ps2 = (5,2)
    using less.IH[of m-5] knights-path-non-nil by auto

  have 27 < length ?ps1 last (take 27 ?ps1) = (2,4) hd (drop 27 ?ps1) = (4,5)
by eval+
  then have step-in ?ps1 (2,4) (4,5)
    unfolding step-in-def using zero-less-numeral by blast
  then have step-in ?ps1 (2,4) (4,5)
    valid-step (2,4) (1,int 5+1)
    valid-step (5,int 5+2) (4,5)
    unfolding valid-step-def by auto
  then show ?thesis
    using ⟨5 ≤ m-5⟩ ps1-prems ps2-IH knights-path-split-concat[of 6 5 ?ps1 m-5
ps2] by auto
qed
qed

```

For every $6 \times m$ -board with $m \geq 5$ there exists a knight's circuit.

lemma *knights-circuit-6xm-exists*:

```

assumes m ≥ 5
shows ∃ ps. knights-circuit (board 6 m) ps
using assms
proof -
  have m ∈ {5,6,7,8,9} ∨ 5 ≤ m-5 using assms by auto
  then show ?thesis
  proof (elim disjE)
    assume m ∈ {5,6,7,8,9}
    then show ?thesis using kc-6xm by fastforce
  next
    let ?ps1=rev kc6x5
    have knights-circuit b6x5 ?ps1 last ?ps1 = (1,1)
      using kc-6xm knights-circuit-rev by (auto simp: last-rev)
    then have ps1-prems: knights-path b6x5 ?ps1 valid-step (last ?ps1) (hd ?ps1)
      unfolding knights-circuit-def using valid-step-rev by auto
    assume m-ge: 5 ≤ m-5
    then obtain ps2 where ps2-prems: knights-path (board 6 (m-5)) ps2 hd ps2
      = (1,1)
      last ps2 = (5,2)
      using knights-path-6xm-ul-exists[of (m-5)] knights-path-non-nil by auto

    have 2 < length ?ps1 last (take 2 ?ps1) = (2,4) hd (drop 2 ?ps1) = (4,5) by

```

```

eval+
  then have step-in ?ps1 (2,4) (4,5)
    unfolding step-in-def using zero-less-numeral by blast
  then have step-in ?ps1 (2,4) (4,5)
    valid-step (2,4) (1,int 5+1)
    valid-step (5,int 5+2) (4,5)
    unfolding valid-step-def by auto
  then have  $\exists ps. \text{knights-path (board 6 } m) ps \wedge \text{hd } ps = \text{hd } ?ps_1 \wedge \text{last } ps = \text{last } ?ps_1$ 
    using m-ge ps1-prems ps2-prems knights-path-split-concat[of 6 5 ?ps1 m-5 ps2] by auto
  then show ?thesis using ps1-prems by (auto simp: knights-circuit-def)
qed

```

$5 \leq ?m \implies \exists ps. \text{knights-path (board 6 } ?m) ps \wedge \text{hd } ps = (1, 1) \wedge \text{last } ps = (5, 2)$ and $5 \leq ?m \implies \exists ps. \text{knights-circuit (board 6 } ?m) ps$ formalize Lemma 2 from [?].

lemmas *knights-path-6xm-exists = knights-path-6xm-ul-exists knights-circuit-6xm-exists*

9 Paths and Circuits for $8 \times m$ -Boards

abbreviation $b8x5 \equiv \text{board } 8 \ 5$

A Knight's path for the (8×5) -board that starts in the lower-left and ends in the upper-left.

28	7	22	39	26
23	40	27	6	21
8	29	38	25	14
37	24	15	20	5
16	9	30	13	34
31	36	33	4	19
10	17	2	35	12
1	32	11	18	3

abbreviation $kp8x5ul \equiv \text{the (to-path}$

$[[28,7,22,39,26],$
 $[23,40,27,6,21],$
 $[8,29,38,25,14],$
 $[37,24,15,20,5],$
 $[16,9,30,13,34],$
 $[31,36,33,4,19],$
 $[10,17,2,35,12],$
 $[1,32,11,18,3]]$

lemma *kp-8x5-ul: knights-path b8x5 kp8x5ul*
by (simp only: knights-path-exec-simp) eval

lemma *kp-8x5-ul-hd*: $hd\ kp8x5ul = (1,1)$ **by** *eval*

lemma *kp-8x5-ul-last*: $last\ kp8x5ul = (7,2)$ **by** *eval*

lemma *kp-8x5-ul-non-nil*: $kp8x5ul \neq []$ **by** *eval*

A Knight's circuit for the (8×5) -board.

26	7	28	15	24
31	16	25	6	29
8	27	30	23	14
17	32	39	34	5
38	9	18	13	22
19	40	33	4	35
10	37	2	21	12
1	20	11	36	3

abbreviation *kc8x5* \equiv *the (to-path*

$[[26,7,28,15,24],$
 $[31,16,25,6,29],$
 $[8,27,30,23,14],$
 $[17,32,39,34,5],$
 $[38,9,18,13,22],$
 $[19,40,33,4,35],$
 $[10,37,2,21,12],$
 $[1,20,11,36,3]]$)

lemma *kc-8x5*: *knights-circuit b8x5 kc8x5*
by (*simp only: knights-circuit-exec-simp*) *eval*

lemma *kc-8x5-hd*: $hd\ kc8x5 = (1,1)$ **by** *eval*

lemma *kc-8x5-last*: $last\ kc8x5 = (3,2)$ **by** *eval*

lemma *kc-8x5-non-nil*: $kc8x5 \neq []$ **by** *eval*

lemma *kc-8x5-si*: *step-in kc8x5 (2,4) (4,5) (is step-in ?ps - -)*

proof –

have $0 < (21::nat)\ 21 < length\ ?ps\ last\ (take\ 21\ ?ps) = (2,4)\ hd\ (drop\ 21\ ?ps)$
 $= (4,5)$

by *eval+*

then show *?thesis unfolding step-in-def by blast*

qed

abbreviation *b8x6* \equiv *board 8 6*

A Knight's path for the (8×6) -board that starts in the lower-left and ends in the upper-left.

42	11	26	9	34	13
25	48	43	12	27	8
44	41	10	33	14	35
47	24	45	20	7	28
40	19	32	3	36	15
23	46	21	6	29	4
18	39	2	31	16	37
1	22	17	38	5	30

abbreviation $kp8x6ul \equiv$ the (to-path

[[42,11,26,9,34,13],
 [25,48,43,12,27,8],
 [44,41,10,33,14,35],
 [47,24,45,20,7,28],
 [40,19,32,3,36,15],
 [23,46,21,6,29,4],
 [18,39,2,31,16,37],
 [1,22,17,38,5,30]])

lemma $kp\text{-}8x6\text{-}ul$: *knight's-path* $b8x6$ $kp8x6ul$
 by (simp only: *knight's-path-exec-simp*) eval

lemma $kp\text{-}8x6\text{-}ul\text{-}hd$: $hd\ kp8x6ul = (1,1)$ by eval

lemma $kp\text{-}8x6\text{-}ul\text{-}last$: $last\ kp8x6ul = (7,2)$ by eval

lemma $kp\text{-}8x6\text{-}ul\text{-}non\text{-}nil$: $kp8x6ul \neq []$ by eval

A Knight's circuit for the (8×6) -board. I have reversed circuit s.t. the circuit steps from $(2,5)$ to $(4,6)$ and not the other way around. This makes the proofs easier.

8	29	24	45	12	37
25	46	9	38	23	44
30	7	28	13	36	11
47	26	39	10	43	22
6	31	4	27	14	35
3	48	17	40	21	42
32	5	2	19	34	15
1	18	33	16	41	20

abbreviation $kc8x6 \equiv$ the (to-path

[[8,29,24,45,12,37],
 [25,46,9,38,23,44],
 [30,7,28,13,36,11],
 [47,26,39,10,43,22],
 [6,31,4,27,14,35],

```

[3,48,17,40,21,42],
[32,5,2,19,34,15],
[1,18,33,16,41,20]])
lemma kc-8x6: knights-circuit b8x6 kc8x6
  by (simp only: knights-circuit-exec-simp) eval

lemma kc-8x6-hd: hd kc8x6 = (1,1) by eval

lemma kc-8x6-non-nil: kc8x6 ≠ [] by eval

lemma kc-8x6-si: step-in kc8x6 (2,5) (4,6) (is step-in ?ps -)
proof -
  have  $0 < (34::nat)$   $34 < \text{length } ?ps$ 
     $\text{last } (\text{take } 34 \text{ } ?ps) = (2,5)$   $\text{hd } (\text{drop } 34 \text{ } ?ps) = (4,6)$  by eval+
  then show ?thesis unfolding step-in-def by blast
qed

```

abbreviation $b8x7 \equiv \text{board } 8 \ 7$

A Knight's path for the (8×7) -board that starts in the lower-left and ends in the upper-left.

38	19	6	55	46	21	8
5	56	39	20	7	54	45
18	37	4	47	34	9	22
3	48	35	40	53	44	33
36	17	52	49	32	23	10
51	2	29	14	41	26	43
16	13	50	31	28	11	24
1	30	15	12	25	42	27

abbreviation $kp8x7ul \equiv \text{the } (to\text{-path}$
 $[[38,19,6,55,46,21,8],$
 $[5,56,39,20,7,54,45],$
 $[18,37,4,47,34,9,22],$
 $[3,48,35,40,53,44,33],$
 $[36,17,52,49,32,23,10],$
 $[51,2,29,14,41,26,43],$
 $[16,13,50,31,28,11,24],$
 $[1,30,15,12,25,42,27]])$
lemma *kp-8x7-ul: knights-path b8x7 kp8x7ul*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-8x7-ul-hd: hd kp8x7ul = (1,1)* **by** *eval*

lemma *kp-8x7-ul-last: last kp8x7ul = (7,2)* **by** *eval*

lemma *kp-8x7-ul-non-nil: kp8x7ul ≠ []* **by** *eval*

A Knight's circuit for the (8×7) -board. I have reversed circuit s.t. the circuit steps from $(2,6)$ to $(4,7)$ and not the other way around. This makes the proofs easier.

36	31	18	53	20	29	44
17	54	35	30	45	52	21
32	37	46	19	8	43	28
55	16	7	34	27	22	51
38	33	26	47	6	9	42
3	56	15	12	25	50	23
14	39	2	5	48	41	10
1	4	13	40	11	24	49

abbreviation $kc8x7 \equiv$ the (to-path

```

[[36,31,18,53,20,29,44],
 [17,54,35,30,45,52,21],
 [32,37,46,19,8,43,28],
 [55,16,7,34,27,22,51],
 [38,33,26,47,6,9,42],
 [3,56,15,12,25,50,23],
 [14,39,2,5,48,41,10],
 [1,4,13,40,11,24,49]]

```

lemma $kc\text{-}8x7$: knights-circuit $b8x7$ $kc8x7$

by (simp only: knights-circuit-exec-simp) eval

lemma $kc\text{-}8x7\text{-}hd$: $hd\ kc8x7 = (1,1)$ **by** eval

lemma $kc\text{-}8x7\text{-}non\text{-}nil$: $kc8x7 \neq []$ **by** eval

lemma $kc\text{-}8x7\text{-}si$: step-in $kc8x7$ $(2,6)$ $(4,7)$ (is step-in ?ps - -)

proof -

have $0 < (41::nat)$ $41 < length\ ?ps$

$last\ (take\ 41\ ?ps) = (2,6)$ $hd\ (drop\ 41\ ?ps) = (4,7)$ **by** eval+

then show ?thesis **unfolding** step-in-def **by** blast

qed

abbreviation $b8x8 \equiv board\ 8\ 8$

The path given for the 8×8 -board that ends in the upper-left is wrong. The Knight cannot move from square 27 to square 28.

24	11	37	9	26	21	39	7
36	64	24	22	38	8	27	20
12	23	10	53	58	49	6	28
63	35	61	50	55	52	19	40
46	13	54	57	48	59	29	5
34	62	47	60	51	56	41	18
14	45	2	32	16	43	4	30
1	33	15	44	3	31	17	42

abbreviation *kp8x8ul-wrong* \equiv the (to-path

[[24,11,37,9,26,21,39,7],
 [36,64,25,22,38,8,27,20],
 [12,23,10,53,58,49,6,28],
 [63,35,61,50,55,52,19,40],
 [46,13,54,57,48,59,29,5],
 [34,62,47,60,51,56,41,18],
 [14,45,2,32,16,43,4,30],
 [1,33,15,44,3,31,17,42]])

value *path-checker* (board-exec 8 8) *kp8x8ul-wrong*

I have computed a correct Knight's path for the 8×8 -board that ends in the upper-left.

38	41	36	27	32	43	20	25
35	64	39	42	21	26	29	44
40	37	6	33	28	31	24	19
5	34	63	14	7	22	45	30
62	13	4	9	58	49	18	23
3	10	61	52	15	8	57	46
12	53	2	59	48	55	50	17
1	60	11	54	51	16	47	56

abbreviation *kp8x8ul* \equiv the (to-path

[[38,41,36,27,32,43,20,25],
 [35,64,39,42,21,26,29,44],
 [40,37,6,33,28,31,24,19],
 [5,34,63,14,7,22,45,30],
 [62,13,4,9,58,49,18,23],
 [3,10,61,52,15,8,57,46],
 [12,53,2,59,48,55,50,17],
 [1,60,11,54,51,16,47,56]])

lemma *kp-8x8-ul: knights-path b8x8 kp8x8ul*
by (*simp only: knights-path-exec-simp*) *eval*

lemma *kp-8x8-ul-hd*: *hd kp8x8ul = (1,1)* **by** *eval*

lemma *kp-8x8-ul-last*: *last kp8x8ul = (7,2)* **by** *eval*

lemma *kp-8x8-ul-non-nil*: *kp8x8ul ≠ []* **by** *eval*

A Knight's circuit for the (8×8) -board.

48	13	30	9	56	45	28	7
31	10	47	50	29	8	57	44
14	49	12	55	46	59	6	27
11	32	37	60	51	54	43	58
36	15	52	63	38	61	26	5
33	64	35	18	53	40	23	42
16	19	2	39	62	21	4	25
1	34	17	20	3	24	41	22

abbreviation *kc8x8* \equiv *the (to-path*

[[48,13,30,9,56,45,28,7],
[31,10,47,50,29,8,57,44],
[14,49,12,55,46,59,6,27],
[11,32,37,60,51,54,43,58],
[36,15,52,63,38,61,26,5],
[33,64,35,18,53,40,23,42],
[16,19,2,39,62,21,4,25],
[1,34,17,20,3,24,41,22]])

lemma *kc-8x8*: *knights-circuit b8x8 kc8x8*
by (*simp only: knights-circuit-exec-simp*) *eval*

lemma *kc-8x8-hd*: *hd kc8x8 = (1,1)* **by** *eval*

lemma *kc-8x8-non-nil*: *kc8x8 ≠ []* **by** *eval*

lemma *kc-8x8-si*: *step-in kc8x8 (2,7) (4,8) (is step-in ?ps -)*

proof –

have $0 < (4 :: nat)$ $4 < \text{length } ?ps$

last (take 4 ?ps) = (2,7) *hd (drop 4 ?ps) = (4,8)* **by** *eval+*

then show *?thesis* **unfolding** *step-in-def* **by** *blast*

qed

abbreviation *b8x9* \equiv *board 8 9*

A Knight's path for the (8×9) -board that starts in the lower-left and ends in the upper-left.

32	47	6	71	30	45	8	43	26
5	72	31	46	7	70	27	22	9
48	33	4	29	64	23	44	25	42
3	60	35	62	69	28	41	10	21
34	49	68	65	36	63	24	55	40
59	2	61	16	67	56	37	20	11
50	15	66	57	52	13	18	39	54
1	58	51	14	17	38	53	12	19

abbreviation $kp8x9ul \equiv$ the (to-path

$[[32,47,6,71,30,45,8,43,26],$
 $[5,72,31,46,7,70,27,22,9],$
 $[48,33,4,29,64,23,44,25,42],$
 $[3,60,35,62,69,28,41,10,21],$
 $[34,49,68,65,36,63,24,55,40],$
 $[59,2,61,16,67,56,37,20,11],$
 $[50,15,66,57,52,13,18,39,54],$
 $[1,58,51,14,17,38,53,12,19]]$)

lemma $kp-8x9-ul$: *knight's-path* $b8x9$ $kp8x9ul$
by (*simp only: knight's-path-exec-simp*) *eval*

lemma $kp-8x9-ul-hd$: $hd\ kp8x9ul = (1,1)$ **by** *eval*

lemma $kp-8x9-ul-last$: $last\ kp8x9ul = (7,2)$ **by** *eval*

lemma $kp-8x9-ul-non-nil$: $kp8x9ul \neq []$ **by** *eval*

A Knight's circuit for the (8×9) -board.

42	19	38	5	36	21	34	7	60
39	4	41	20	63	6	59	22	33
18	43	70	37	58	35	68	61	8
3	40	49	64	69	62	57	32	23
50	17	44	71	48	67	54	9	56
45	2	65	14	27	12	29	24	31
16	51	72	47	66	53	26	55	10
1	46	15	52	13	28	11	30	25

abbreviation $kc8x9 \equiv$ the (to-path

$[[42,19,38,5,36,21,34,7,60],$
 $[39,4,41,20,63,6,59,22,33],$
 $[18,43,70,37,58,35,68,61,8],$
 $[3,40,49,64,69,62,57,32,23],$
 $[50,17,44,71,48,67,54,9,56],$
 $[45,2,65,14,27,12,29,24,31],$
 $[16,51,72,47,66,53,26,55,10],$

```

[1,46,15,52,13,28,11,30,25]])
lemma kc-8x9: knights-circuit b8x9 kc8x9
  by (simp only: knights-circuit-exec-simp) eval

lemma kc-8x9-hd: hd kc8x9 = (1,1) by eval

lemma kc-8x9-non-nil: kc8x9 ≠ [] by eval

lemma kc-8x9-si: step-in kc8x9 (2,8) (4,9) (is step-in ?ps -)
proof -
  have  $0 < (55::nat)$   $55 < \text{length } ?ps$ 
     $\text{last } (\text{take } 55 ?ps) = (2,8)$   $\text{hd } (\text{drop } 55 ?ps) = (4,9)$  by eval+
  then show ?thesis unfolding step-in-def by blast
qed

lemmas kp-8xm-ul =
  kp-8x5-ul kp-8x5-ul-hd kp-8x5-ul-last kp-8x5-ul-non-nil
  kp-8x6-ul kp-8x6-ul-hd kp-8x6-ul-last kp-8x6-ul-non-nil
  kp-8x7-ul kp-8x7-ul-hd kp-8x7-ul-last kp-8x7-ul-non-nil
  kp-8x8-ul kp-8x8-ul-hd kp-8x8-ul-last kp-8x8-ul-non-nil
  kp-8x9-ul kp-8x9-ul-hd kp-8x9-ul-last kp-8x9-ul-non-nil

lemmas kc-8xm =
  kc-8x5 kc-8x5-hd kc-8x5-last kc-8x5-non-nil kc-8x5-si
  kc-8x6 kc-8x6-hd kc-8x6-non-nil kc-8x6-si
  kc-8x7 kc-8x7-hd kc-8x7-non-nil kc-8x7-si
  kc-8x8 kc-8x8-hd kc-8x8-non-nil kc-8x8-si
  kc-8x9 kc-8x9-hd kc-8x9-non-nil kc-8x9-si

For every  $8 \times m$ -board with  $m \geq 5$  there exists a knight's circuit.

lemma knights-circuit-8xm-exists:
  assumes  $m \geq 5$ 
  shows  $\exists ps. \text{knights-circuit } (\text{board } 8 \ m) \ ps \wedge \text{step-in } ps \ (2, \text{int } m-1) \ (4, \text{int } m)$ 
  using assms
proof (induction m rule: less-induct)
  case (less m)
  then have  $m \in \{5,6,7,8,9\} \vee 5 \leq m-5$  by auto
  then show ?case
  proof (elim disjE)
    assume  $m \in \{5,6,7,8,9\}$ 
    then show ?thesis using kc-8xm by fastforce
  next
    let  $?ps_2 = \text{kc8x5}$ 
    let  $?b_2 = \text{board } 8 \ 5$ 
    have  $ps_2\text{-prems: knights-circuit } ?b_2 \ ?ps_2 \ \text{hd } ?ps_2 = (1,1) \ \text{last } ?ps_2 = (3,2)$ 
      using kc-8xm by auto
    have  $21 < \text{length } ?ps_2$   $\text{last } (\text{take } 21 ?ps_2) = (2, \text{int } 5-1) \ \text{hd } (\text{drop } 21 ?ps_2) =$ 
       $(4, \text{int } 5)$ 
      by eval+

```

```

then have si: step-in ?ps2 (2,int 5-1) (4,int 5)
  unfolding step-in-def using zero-less-numeral by blast
assume m-ge: 5 ≤ m-5
then obtain ps1 where ps1-IH: knights-circuit (board 8 (m-5)) ps1
  step-in ps1 (2,int (m-5)-1) (4,int (m-5))
  using less.IH[of m-5] knights-path-non-nil by auto
then show ?thesis
  using m-ge ps2-prems si knights-circuit-lr-concat[of 8 m-5 ps1 5 ?ps2] by
auto
qed
qed

```

For every $8 \times m$ -board with $m \geq 5$ there exists a knight's path that starts in $(1,1)$ (bottom-left) and ends in $(7,2)$ (top-left).

lemma *knights-path-8xm-ul-exists*:

```

assumes m ≥ 5
shows ∃ ps. knights-path (board 8 m) ps ∧ hd ps = (1,1) ∧ last ps = (7,2)
using assms
proof -
  have m ∈ {5,6,7,8,9} ∨ 5 ≤ m-5 using assms by auto
  then show ?thesis
  proof (elim disjE)
    assume m ∈ {5,6,7,8,9}
    then show ?thesis using kp-8xm-ul by fastforce
  next
    let ?ps1=kp8x5ul
    have ps1-prems: knights-path b8x5 ?ps1 hd ?ps1 = (1,1) last ?ps1 = (7,2)
      using kp-8xm-ul by auto
    assume m-ge: 5 ≤ m-5
    then have b-prems: 5 ≤ min 8 (m-5)
      unfolding board-def by auto

    obtain ps2 where knights-circuit (board 8 (m-5)) ps2
      using m-ge knights-circuit-8xm-exists[of (m-5)] knights-path-non-nil by auto
    then obtain ps2' where ps2'-prems': knights-circuit (board 8 (m-5)) ps2'
      hd ps2' = (1,1) last ps2' = (3,2)
      using b-prems ⟨5 ≤ min 8 (m-5)⟩ rotate-knights-circuit by blast
    then have ps2'-path: knights-path (board 8 (m-5)) (rev ps2')
      valid-step (last ps2') (hd ps2') hd (rev ps2') = (3,2) last (rev ps2') = (1,1)
      unfolding knights-circuit-def using knights-path-rev by (auto simp: hd-rev
last-rev)

    have 34 < length ?ps1 last (take 34 ?ps1) = (4,5) hd (drop 34 ?ps1) = (2,4)
  by eval+
  then have step-in ?ps1 (4,5) (2,4)
    unfolding step-in-def using zero-less-numeral by blast
  then have step-in ?ps1 (4,5) (2,4)
    valid-step (4,5) (3,int 5+2)
    valid-step (1,int 5+1) (2,4)

```

```

unfolding valid-step-def by auto
then have  $\exists ps. \text{knights-path } (\text{board } 8 \ m) \ ps \wedge \text{hd } ps = \text{hd } ?ps_1 \wedge \text{last } ps = \text{last } ?ps_1$ 
using m-ge ps1-prems ps2'-prems' ps2'-path
knights-path-split-concat[of 8 5 ?ps1 m-5 rev ps2] by auto
then show ?thesis using ps1-prems by auto
qed
qed

```

$5 \leq ?m \implies \exists ps. \text{knights-circuit } (\text{board } 8 \ ?m) \ ps \wedge \text{step-in } ps \ (2, \text{int } ?m - 1) \ (4, \text{int } ?m)$ and $5 \leq ?m \implies \exists ps. \text{knights-path } (\text{board } 8 \ ?m) \ ps \wedge \text{hd } ps = (1, 1) \wedge \text{last } ps = (7, 2)$ formalize Lemma 3 from [?].

lemmas *knights-path-8xm-exists = knights-circuit-8xm-exists knights-path-8xm-ul-exists*

10 Paths and Circuits for $n \times m$ -Boards

In this section the desired theorems are proved. The proof uses the previous lemmas to construct paths and circuits for arbitrary $n \times m$ -boards.

A Knight's path for the (5×5) -board that starts in the lower-left and ends in the upper-left.

7	20	9	14	5
10	25	6	21	16
19	8	15	4	13
24	11	2	17	22
1	18	23	12	3

abbreviation *kp5x5ul* \equiv *the (to-path*

[[7,20,9,14,5],
[10,25,6,21,16],
[19,8,15,4,13],
[24,11,2,17,22],
[1,18,23,12,3]])

lemma *kp-5x5-ul: knights-path b5x5 kp5x5ul*

by (*simp only: knights-path-exec-simp*) *eval*

A Knight's path for the (5×7) -board that starts in the lower-left and ends in the upper-left.

17	14	25	6	19	8	29
26	35	18	15	28	5	20
13	16	27	24	7	30	9
34	23	2	11	32	21	4
1	12	33	22	3	10	31

abbreviation $kp5x7ul \equiv$ the (to-path

$[[17,14,25,6,19,8,29],$
 $[26,35,18,15,28,5,20],$
 $[13,16,27,24,7,30,9],$
 $[34,23,2,11,32,21,4],$
 $[1,12,33,22,3,10,31]])$

lemma $kp-5x7-ul$: knights-path $b5x7$ $kp5x7ul$

by (simp only: knights-path-exec-simp) eval

A Knight's path for the (5×9) -board that starts in the lower-left and ends in the upper-left.

7	12	37	42	5	18	23	32	27
38	45	6	11	36	31	26	19	24
13	8	43	4	41	22	17	28	33
44	39	2	15	10	35	30	25	20
1	14	9	40	3	16	21	34	29

abbreviation $kp5x9ul \equiv$ the (to-path

$[[7,12,37,42,5,18,23,32,27],$
 $[38,45,6,11,36,31,26,19,24],$
 $[13,8,43,4,41,22,17,28,33],$
 $[44,39,2,15,10,35,30,25,20],$
 $[1,14,9,40,3,16,21,34,29]])$

lemma $kp-5x9-ul$: knights-path $b5x9$ $kp5x9ul$

by (simp only: knights-path-exec-simp) eval

abbreviation $b7x7 \equiv$ board $7\ 7$

A Knight's path for the (7×7) -board that starts in the lower-left and ends in the upper-left.

9	30	19	42	7	32	17
20	49	8	31	18	43	6
29	10	41	36	39	16	33
48	21	38	27	34	5	44
11	28	35	40	37	26	15
22	47	2	13	24	45	4
1	12	23	46	3	14	25

abbreviation $kp7x7ul \equiv$ the (to-path

$[[9,30,19,42,7,32,17],$
 $[20,49,8,31,18,43,6],$
 $[29,10,41,36,39,16,33],$
 $[48,21,38,27,34,5,44],$
 $[11,28,35,40,37,26,15],$
 $[22,47,2,13,24,45,4],$

[1,12,23,46,3,14,25]])
lemma *kp-7x7-ul: knights-path b7x7 kp7x7ul*
by (*simp only: knights-path-exec-simp*) *eval*

abbreviation *b7x9* \equiv *board 7 9*

A Knight's path for the (7×9) -board that starts in the lower-left and ends in the upper-left.

59	4	17	50	37	6	19	30	39
16	63	58	5	18	51	38	7	20
3	60	49	36	57	42	29	40	31
48	15	62	43	52	35	56	21	8
61	2	13	26	45	28	41	32	55
14	47	44	11	24	53	34	9	22
1	12	25	46	27	10	23	54	33

abbreviation *kp7x9ul* \equiv *the (to-path*
[[59,4,17,50,37,6,19,30,39],
[16,63,58,5,18,51,38,7,20],
[3,60,49,36,57,42,29,40,31],
[48,15,62,43,52,35,56,21,8],
[61,2,13,26,45,28,41,32,55],
[14,47,44,11,24,53,34,9,22],
[1,12,25,46,27,10,23,54,33]])

lemma *kp-7x9-ul: knights-path b7x9 kp7x9ul*
by (*simp only: knights-path-exec-simp*) *eval*

abbreviation *b9x7* \equiv *board 9 7*

A Knight's path for the (9×7) -board that starts in the lower-left and ends in the upper-left.

5	20	53	48	7	22	31
52	63	6	21	32	55	8
19	4	49	54	47	30	23
62	51	46	33	56	9	58
3	18	61	50	59	24	29
14	43	34	45	28	57	10
17	2	15	60	35	38	25
42	13	44	27	40	11	36
1	16	41	12	37	26	39

abbreviation *kp9x7ul* \equiv *the (to-path*
[[5,20,53,48,7,22,31],
[52,63,6,21,32,55,8],
[19,4,49,54,47,30,23],
[62,51,46,33,56,9,58],
[3,18,61,50,59,24,29],
[14,43,34,45,28,57,10],
[17,2,15,60,35,38,25],
[42,13,44,27,40,11,36],
[1,16,41,12,37,26,39]])


```

[19,4,49,54,47,30,23],
[62,51,46,33,56,9,58],
[3,18,61,50,59,24,29],
[14,43,34,45,28,57,10],
[17,2,15,60,35,38,25],
[42,13,44,27,40,11,36],
[1,16,41,12,37,26,39]])
lemma kp-9x7-ul: knights-path b9x7 kp9x7ul
by (simp only: knights-path-exec-simp) eval

```

abbreviation *b9x9* \equiv *board 9 9*

A Knight's path for the (9×9) -board that starts in the lower-left and ends in the upper-left.

13	26	39	52	11	24	37	50	9
40	81	12	25	38	51	10	23	36
27	14	53	58	63	68	73	8	49
80	41	64	67	72	57	62	35	22
15	28	59	54	65	74	69	48	7
42	79	66	71	76	61	56	21	34
29	16	77	60	55	70	75	6	47
78	43	2	31	18	45	4	33	20
1	30	17	44	3	32	19	46	5

```

abbreviation kp9x9ul  $\equiv$  the (to-path
[[13,26,39,52,11,24,37,50,9],
[40,81,12,25,38,51,10,23,36],
[27,14,53,58,63,68,73,8,49],
[80,41,64,67,72,57,62,35,22],
[15,28,59,54,65,74,69,48,7],
[42,79,66,71,76,61,56,21,34],
[29,16,77,60,55,70,75,6,47],
[78,43,2,31,18,45,4,33,20],
[1,30,17,44,3,32,19,46,5]])
lemma kp-9x9-ul: knights-path b9x9 kp9x9ul
by (simp only: knights-path-exec-simp) eval

```

The following lemma is a sub-proof used in Lemma 4 in [?]. I moved the sub-proof out to a separate lemma.

```

lemma knights-circuit-exists-even-n-gr10:
assumes even n n ≥ 10 m ≥ 5
   $\exists ps. \text{knights-path (board (n-5) m) ps} \wedge \text{hd ps} = (\text{int (n-5)}, 1)$ 
   $\wedge \text{last ps} = (\text{int (n-5)}-1, \text{int m}-1)$ 
shows  $\exists ps. \text{knights-circuit (board m n) ps}$ 
using assms
proof –

```

```

let ?b2=board (n-5) m
assume n ≥ 10
then obtain ps2 where ps2-prems: knights-path ?b2 ps2 hd ps2 = (int (n-5),1)

    last ps2 = (int (n-5)-1,int m-1)
    using assms by auto
let ?ps2-m2=mirror2 ps2
have ps2-m2-prems: knights-path ?b2 ?ps2-m2 hd ?ps2-m2 = (int (n-5),int m)
    last ?ps2-m2 = (int (n-5)-1,2)
    using ps2-prems mirror2-knights-path hd-mirror2 last-mirror2 by auto

obtain ps1 where ps1-prems: knights-path (board 5 m) ps1 hd ps1 = (1,1)last
ps1 = (2,int m-1)
    using assms knights-path-5xm-exists by auto
let ?ps1'=trans-path (int (n-5),0) ps1
let ?b1'=trans-board (int (n-5),0) (board 5 m)
have ps1'-prems: knights-path ?b1' ?ps1' hd ?ps1' = (int (n-5)+1,1)
    last ?ps1' = (int (n-5)+2,int m-1)
    using ps1-prems trans-knights-path knights-path-non-nil hd-trans-path last-trans-path
by auto

let ?ps=?ps1'@?ps2-m2
let ?psT=transpose ?ps

have n-5 ≥ 5 using ⟨n ≥ 10⟩ by auto
have inter: ?b1' ∩ ?b2 = {}
    unfolding trans-board-def board-def using ⟨n-5 ≥ 5⟩ by auto
have union: ?b1' ∪ ?b2 = board n m
    using ⟨n-5 ≥ 5⟩ board-concatT[of n-5 m 5] by auto

have vs: valid-step (last ?ps1') (hd ?ps2-m2) and valid-step (last ?ps2-m2) (hd
?ps1')
    unfolding valid-step-def using ps1'-prems ps2-m2-prems by auto
then have vs-c: valid-step (last ?ps) (hd ?ps)
    using ps1'-prems ps2-m2-prems knights-path-non-nil by auto

have knights-path (board n m) ?ps
    using ps1'-prems ps2-m2-prems inter vs union knights-path-append[of ?b1' ?ps1'
?b2 ?ps2-m2]
    by auto
then have knights-circuit (board n m) ?ps
    unfolding knights-circuit-def using vs-c by auto
then show ?thesis using transpose-knights-circuit by auto
qed

```

For every $n \times m$ -board with $\min n m \geq 5$ and odd n there exists a Knight's path that starts in $(n,1)$ (top-left) and ends in $(n-1,m-1)$ (top-right).

This lemma formalizes Lemma 4 from [?]. Formalizing the proof of this lemma was quite challenging as a lot of details on how to exactly combine

the boards are left out in the original proof in [?].

lemma *knight-path-odd-n-exists*:

assumes $\text{odd } n \text{ min } n \ m \geq 5$

shows $\exists ps. \text{knight-path } (\text{board } n \ m) \ ps \wedge \text{hd } ps = (\text{int } n, 1) \wedge \text{last } ps = (\text{int } n-1, \text{int } m-1)$

using *assms*

proof –

obtain x **where** $x = n + m$ **by** *auto*

then show *?thesis*

using *assms*

proof (*induction x arbitrary: n m rule: less-induct*)

case (*less x*)

then have $m = 5 \vee m = 6 \vee m = 7 \vee m = 8 \vee m = 9 \vee m \geq 10$ **by** *auto*

then show *?case*

proof (*elim disjE*)

assume [*simp*]: $m = 5$

have $\text{odd } n \ n \geq 5$ **using** *less* **by** *auto*

then have $n = 5 \vee n = 7 \vee n = 9 \vee n-5 \geq 5$ **by** *presburger*

then show *?thesis*

proof (*elim disjE*)

assume [*simp*]: $n = 5$

let $?ps = \text{mirror1 } (\text{transpose } kp5x5ul)$

have $kp: \text{knight-path } (\text{board } n \ m) \ ?ps$

using *kp-5x5-ul rot90-knight-path* **by** *auto*

have $\text{hd } ?ps = (\text{int } n, 1) \ \text{last } ?ps = (\text{int } n-1, \text{int } m-1)$

by (*simp only: <m = 5> <n = 5> | eval*)**+**

then show *?thesis* **using** kp **by** *auto*

next

assume [*simp*]: $n = 7$

let $?ps = \text{mirror1 } (\text{transpose } kp5x7ul)$

have $kp: \text{knight-path } (\text{board } n \ m) \ ?ps$

using *kp-5x7-ul rot90-knight-path* **by** *auto*

have $\text{hd } ?ps = (\text{int } n, 1) \ \text{last } ?ps = (\text{int } n-1, \text{int } m-1)$

by (*simp only: <m = 5> <n = 7> | eval*)**+**

then show *?thesis* **using** kp **by** *auto*

next

assume [*simp*]: $n = 9$

let $?ps = \text{mirror1 } (\text{transpose } kp5x9ul)$

have $kp: \text{knight-path } (\text{board } n \ m) \ ?ps$

using *kp-5x9-ul rot90-knight-path* **by** *auto*

have $\text{hd } ?ps = (\text{int } n, 1) \ \text{last } ?ps = (\text{int } n-1, \text{int } m-1)$

by (*simp only: <m = 5> <n = 9> | eval*)**+**

then show *?thesis* **using** kp **by** *auto*

next

let $?b_2 = \text{board } m \ (n-5)$

assume $n-5 \geq 5$

then have $\exists ps. \text{knight-circuit } ?b_2 \ ps$

proof –

have $n-5 = 6 \vee n-5 = 8 \vee n-5 \geq 10$

```

    using  $\langle n-5 \geq 5 \rangle$  less by presburger
  then show ?thesis
proof (elim disjE)
  assume  $n-5 = 6$ 
  then obtain ps where knights-circuit (board (n-5) m) ps
    using knights-path-6xm-exists[of m] by auto
  then show ?thesis
    using transpose-knights-circuit by auto
next
  assume  $n-5 = 8$ 
  then obtain ps where knights-circuit (board (n-5) m) ps
    using knights-path-8xm-exists[of m] by auto
  then show ?thesis
    using transpose-knights-circuit by auto
next
  assume  $n-5 \geq 10$ 
  then show ?thesis
    using less less.IH[of n-10+m n-10 m]
      knights-circuit-exists-even-n-gr10[of n-5 m] by auto
qed
qed
then obtain ps2 where knights-circuit ?b2 ps2 hd ps2 = (1,1) last ps2 =
(3,2)
  using  $\langle n-5 \geq 5 \rangle$  rotate-knights-circuit[of m n-5] by auto
then have rev-ps2-prems: knights-path ?b2 (rev ps2) valid-step (last ps2) (hd
ps2)
  hd (rev ps2) = (3,2) last (rev ps2) = (1,1)
  unfolding knights-circuit-def using knights-path-rev by (auto simp: hd-rev
last-rev)

let ?ps1=kp5x5ul
have ps1-prems: knights-path (board 5 5) ?ps1 hd ?ps1 = (1,1) last ?ps1 =
(4,2)
  using kp-5x5-ul by simp eval+

  have 16 < length ?ps1 last (take 16 ?ps1) = (4,5) hd (drop 16 ?ps1) =
(2,4) by eval+
  then have si: step-in ?ps1 (4,5) (2,4)
    unfolding step-in-def using zero-less-numeral by blast

  have vs: valid-step (4,5) (3,int 5+2) valid-step (1,int 5+1) (2,4)
    unfolding valid-step-def by auto

obtain ps where knights-path (board m n) ps hd ps = (1,1) last ps = (4,2)
  using  $\langle n-5 \geq 5 \rangle$  ps1-prems rev-ps2-prems si vs
    knights-path-split-concat[of 5 5 ?ps1 n-5 rev ps2 (4,5) (2,4)] by auto
then show ?thesis
  using rot90-knights-path hd-rot90-knights-path last-rot90-knights-path by
fastforce

```

```

qed
next
  assume [simp]:  $m = 6$ 
  then obtain  $ps$  where
     $ps$ -prems:  $knights\text{-}path\ (board\ m\ n)\ ps\ hd\ ps = (1,1)\ last\ ps = (int\ m-1,2)$ 
    using  $less\ knights\text{-}path\text{-}6xm\text{-}exists[of\ n]$  by auto
  let  $?ps' = mirror1\ (transpose\ ps)$ 
  have  $knights\text{-}path\ (board\ n\ m)\ ?ps'\ hd\ ?ps' = (int\ n,1)\ last\ ?ps' = (int\ n-1,int\ m-1)$ 
  using  $ps$ -prems  $rot90\text{-}knights\text{-}path\ hd\ rot90\text{-}knights\text{-}path\ last\ rot90\text{-}knights\text{-}path$ 
  by auto
  then show  $?thesis$  by auto
next
  assume [simp]:  $m = 7$ 
  have  $odd\ n\ n \geq 5$  using  $less$  by auto
  then have  $n = 5 \vee n = 7 \vee n = 9 \vee n-5 \geq 5$  by presburger
  then show  $?thesis$ 
  proof (elim disjE)
    assume [simp]:  $n = 5$ 
    let  $?ps = mirror1\ kp5x7lr$ 
    have  $kp$ :  $knights\text{-}path\ (board\ n\ m)\ ?ps$ 
      using  $kp\text{-}5x7\text{-}lr\ mirror1\text{-}knights\text{-}path$  by auto
    have  $hd\ ?ps = (int\ n,1)\ last\ ?ps = (int\ n-1,int\ m-1)$ 
      by (simp only:  $\langle m = 7 \rangle\ \langle n = 5 \rangle\ | eval$ ) +
    then show  $?thesis$  using  $kp$  by auto
  next
    assume [simp]:  $n = 7$ 
    let  $?ps = mirror1\ (transpose\ kp7x7ul)$ 
    have  $kp$ :  $knights\text{-}path\ (board\ n\ m)\ ?ps$ 
      using  $kp\text{-}7x7\text{-}ul\ rot90\text{-}knights\text{-}path$  by auto
    have  $hd\ ?ps = (int\ n,1)\ last\ ?ps = (int\ n-1,int\ m-1)$ 
      by (simp only:  $\langle m = 7 \rangle\ \langle n = 7 \rangle\ | eval$ ) +
    then show  $?thesis$  using  $kp$  by auto
  next
    assume [simp]:  $n = 9$ 
    let  $?ps = mirror1\ (transpose\ kp7x9ul)$ 
    have  $kp$ :  $knights\text{-}path\ (board\ n\ m)\ ?ps$ 
      using  $kp\text{-}7x9\text{-}ul\ rot90\text{-}knights\text{-}path$  by auto
    have  $hd\ ?ps = (int\ n,1)\ last\ ?ps = (int\ n-1,int\ m-1)$ 
      by (simp only:  $\langle m = 7 \rangle\ \langle n = 9 \rangle\ | eval$ ) +
    then show  $?thesis$  using  $kp$  by auto
  next
    let  $?b_2 = board\ m\ (n-5)$ 
    let  $?b_2\ T = board\ (n-5)\ m$ 
    assume  $n-5 \geq 5$ 
    then have  $\exists ps. knights\text{-}circuit\ ?b_2\ ps$ 
    proof -
      have  $n-5 = 6 \vee n-5 = 8 \vee n-5 \geq 10$ 
      using  $\langle n-5 \geq 5 \rangle\ less$  by presburger

```

```

then show ?thesis
proof (elim disjE)
  assume  $n-5 = 6$ 
  then obtain  $ps$  where  $knights-circuit$  (board  $(n-5)$   $m$ )  $ps$ 
    using  $knights-path-6xm-exists[of\ m]$  by auto
  then show ?thesis
    using  $transpose-knights-circuit$  by auto
next
  assume  $n-5 = 8$ 
  then obtain  $ps$  where  $knights-circuit$  (board  $(n-5)$   $m$ )  $ps$ 
    using  $knights-path-8xm-exists[of\ m]$  by auto
  then show ?thesis
    using  $transpose-knights-circuit$  by auto
next
  assume  $n-5 \geq 10$ 
  then show ?thesis
    using  $less\ less.IH[of\ n-10+m\ n-10\ m]$ 
       $knights-circuit-exists-even-n-gr10[of\ n-5\ m]$  by auto
qed
qed
then obtain  $ps_2$  where  $ps_2-prems: knights-circuit\ ?b_2\ ps_2\ hd\ ps_2 = (1,1)$ 
   $last\ ps_2 = (3,2)$ 
  using  $\langle n-5 \geq 5 \rangle rotate-knights-circuit[of\ m\ n-5]$  by auto
let  $?ps_2T = transpose\ ps_2$ 
have  $ps_2T-prems: knights-path\ ?b_2T\ ?ps_2T\ hd\ ?ps_2T = (1,1)\ last\ ?ps_2T =$ 
 $(2,3)$ 
  using  $ps_2-prems\ transpose-knights-path\ knights-path-non-nil\ hd-transpose$ 
 $last-transpose$ 
  unfolding  $knights-circuit-def\ transpose-square-def$  by auto

let  $?ps_1 = kp5x7lr$ 
have  $ps_1-prems: knights-path\ b5x7\ ?ps_1\ hd\ ?ps_1 = (1,1)\ last\ ?ps_1 = (2,6)$ 
  using  $kp-5x7-lr$  by simp eval+

have  $29 < length\ ?ps_1\ last\ (take\ 29\ ?ps_1) = (4,2)\ hd\ (drop\ 29\ ?ps_1) =$ 
 $(5,4)$  by eval+
then have  $si: step-in\ ?ps_1\ (4,2)\ (5,4)$ 
  unfolding  $step-in-def$  using  $zero-less-numeral$  by blast

have  $vs: valid-step\ (4,2)\ (int\ 5+1,1)\ valid-step\ (int\ 5+2,3)\ (5,4)$ 
  unfolding  $valid-step-def$  by auto

obtain  $ps$  where  $knights-path$  (board  $n$   $m$ )  $ps\ hd\ ps = (1,1)\ last\ ps = (2,6)$ 
  using  $\langle n-5 \geq 5 \rangle ps_1-prems\ ps_2T-prems\ si\ vs$ 
   $knights-path-split-concatT[of\ 5\ m\ ?ps_1\ n-5\ ?ps_2T\ (4,2)\ (5,4)]$  by auto
then show ?thesis
  using  $mirror1-knights-path\ hd-mirror1\ last-mirror1$  by fastforce
qed
next

```

```

    assume [simp]: m = 8
    then obtain ps where ps-prems: knights-path (board m n) ps hd ps = (1,1)
      last ps = (int m-1,2)
      using less knights-path-8xm-exists[of n] by auto
    let ?ps'=mirror1 (transpose ps)
    have knights-path (board n m) ?ps' hd ?ps' = (int n,1) last ?ps' = (int n-1,int
m-1)
      using ps-prems rot90-knights-path hd-rot90-knights-path last-rot90-knights-path
by auto
    then show ?thesis by auto
  next
    assume [simp]: m = 9
    have odd n n ≥ 5 using less by auto
    then have n = 5 ∨ n = 7 ∨ n = 9 ∨ n-5 ≥ 5 by presburger
    then show ?thesis
  proof (elim disjE)
    assume [simp]: n = 5
    let ?ps=mirror1 kp5x9lr
    have kp: knights-path (board n m) ?ps
      using kp-5x9-lr mirror1-knights-path by auto
    have hd ?ps = (int n,1) last ?ps = (int n-1,int m-1)
      by (simp only: ⟨m = 9⟩ ⟨n = 5⟩ | eval)+
    then show ?thesis using kp by auto
  next
    assume [simp]: n = 7
    let ?ps=mirror1 (transpose kp9x7ul)
    have kp: knights-path (board n m) ?ps
      using kp-9x7-ul rot90-knights-path by auto
    have hd ?ps = (int n,1) last ?ps = (int n-1,int m-1)
      by (simp only: ⟨m = 9⟩ ⟨n = 7⟩ | eval)+
    then show ?thesis using kp by auto
  next
    assume [simp]: n = 9
    let ?ps=mirror1 (transpose kp9x9ul)
    have kp: knights-path (board n m) ?ps
      using kp-9x9-ul rot90-knights-path by auto
    have hd ?ps = (int n,1) last ?ps = (int n-1,int m-1)
      by (simp only: ⟨m = 9⟩ ⟨n = 9⟩ | eval)+
    then show ?thesis using kp by auto
  next
    let ?b2=board m (n-5)
    let ?b2T=board (n-5) m
    assume n-5 ≥ 5
    then have ∃ ps. knights-circuit ?b2 ps
    proof -
      have n-5 = 6 ∨ n-5 = 8 ∨ n-5 ≥ 10
        using ⟨n-5 ≥ 5⟩ less by presburger
      then show ?thesis
    proof (elim disjE)

```

```

    assume  $n-5 = 6$ 
    then obtain  $ps$  where  $knights-circuit$  (board  $(n-5)$   $m$ )  $ps$ 
      using  $knights-path-6xm-exists[of\ m]$  by auto
    then show  $?thesis$ 
      using  $transpose-knights-circuit$  by auto
  next
    assume  $n-5 = 8$ 
    then obtain  $ps$  where  $knights-circuit$  (board  $(n-5)$   $m$ )  $ps$ 
      using  $knights-path-8xm-exists[of\ m]$  by auto
    then show  $?thesis$ 
      using  $transpose-knights-circuit$  by auto
  next
    assume  $n-5 \geq 10$ 
    then show  $?thesis$ 
      using  $less\ less.IH[of\ n-10+m\ n-10\ m]$ 
         $knights-circuit-exists-even-n-gr10[of\ n-5\ m]$  by auto
  qed
qed
then obtain  $ps_2$  where  $ps_2-prems: knights-circuit\ ?b_2\ ps_2\ hd\ ps_2 = (1,1)$ 
   $last\ ps_2 = (3,2)$ 
  using  $\langle n-5 \geq 5 \rangle rotate-knights-circuit[of\ m\ n-5]$  by auto
let  $?ps_2T = transpose\ (rev\ ps_2)$ 
have  $ps_2T-prems: knights-path\ ?b_2T\ ?ps_2T\ hd\ ?ps_2T = (2,3)\ last\ ?ps_2T =$ 
 $(1,1)$ 
  using  $ps_2-prems\ knights-path-rev\ transpose-knights-path\ knights-path-non-nil$ 
   $hd-transpose\ last-transpose$ 
  unfolding  $knights-circuit-def\ transpose-square-def$  by (auto simp:  $hd-rev$ 
 $last-rev$ )

let  $?ps_1 = kp5x9lr$ 
have  $ps_1-prems: knights-path\ b5x9\ ?ps_1\ hd\ ?ps_1 = (1,1)\ last\ ?ps_1 = (2,8)$ 
  using  $kp-5x9-lr$  by simp eval+

have  $16 < length\ ?ps_1\ last\ (take\ 16\ ?ps_1) = (5,4)\ hd\ (drop\ 16\ ?ps_1) =$ 
 $(4,2)$  by eval+
then have  $si: step-in\ ?ps_1\ (5,4)\ (4,2)$ 
  unfolding  $step-in-def$  using  $zero-less-numeral$  by blast

have  $vs: valid-step\ (5,4)\ (int\ 5+2,3)\ valid-step\ (int\ 5+1,1)\ (4,2)$ 
  unfolding  $valid-step-def$  by auto

obtain  $ps$  where  $knights-path$  (board  $n\ m$ )  $ps\ hd\ ps = (1,1)\ last\ ps = (2,8)$ 
  using  $\langle n-5 \geq 5 \rangle ps_1-prems\ ps_2T-prems\ si\ vs$ 
   $knights-path-split-concatT[of\ 5\ m\ ?ps_1\ n-5\ ?ps_2T\ (5,4)\ (4,2)]$  by auto
then show  $?thesis$ 
  using  $mirror1-knights-path\ hd-mirror1\ last-mirror1$  by fastforce
qed
next

```



```

let ?b1=board n 5
let ?b2=board n (m-5)
assume m ≥ 10
then have n+5 < x 5 ≤ min n 5 n+(m-5) < x 5 ≤ min n (m-5)
  using less by auto
then obtain ps1 ps2 where kp-prems:
  knights-path ?b1 ps1 hd ps1 = (int n,1) last ps1 = (int n-1,4)
  knights-path (board n (m-5)) ps2 hd ps2 = (int n,1) last ps2 = (int n-1,int
(m-5)-1)
  using less.prem less.IH[of n+5 n 5] less.IH[of n+(m-5) n m-5] by auto
let ?ps=ps1@trans-path (0,int 5) ps2
have valid-step (last ps1) (int n,int 5+1)
  unfolding valid-step-def using kp-prems by auto
then have knights-path (board n m) ?ps hd ?ps = (int n,1) last ?ps = (int
n-1,int m-1)
  using ⟨m ≥ 10⟩ kp-prems knights-path-concat[of n 5 ps1 m-5 ps2]
  knights-path-non-nil trans-path-non-nil last-trans-path by auto
then show ?thesis by auto
qed
qed
qed

```

Auxiliary lemma that constructs a Knight's circuit if $m \geq 5$ and $n \geq 10 \wedge$ even n .

```

lemma knights-circuit-exists-n-even-gr-10:
  assumes n ≥ 10 ∧ even n m ≥ 5
  shows ∃ ps. knights-circuit (board n m) ps
  using assms
proof –
  obtain ps1 where ps1-prems: knights-path (board 5 m) ps1 hd ps1 = (1,1)
    last ps1 = (2,int m-1)
  using assms knights-path-5xm-exists by auto
  let ?ps1'=trans-path (int (n-5),0) ps1
  let ?b5xm'=trans-board (int (n-5),0) (board 5 m)
  have ps1'-prems: knights-path ?b5xm' ?ps1' hd ?ps1' = (int (n-5)+1,1)
    last ?ps1' = (int (n-5)+2,int m-1)
  using ps1-prems trans-knights-path knights-path-non-nil hd-trans-path last-trans-path
by auto

  assume n ≥ 10 ∧ even n
  then have odd (n-5) min (n-5) m ≥ 5 using assms by auto
  then obtain ps2 where ps2-prems: knights-path (board (n-5) m) ps2 hd ps2 =
(int (n-5),1)
    last ps2 = (int (n-5)-1,int m-1)
  using knights-path-odd-n-exists[of n-5 m] by auto
  let ?ps2'=mirror2 ps2
  have ps2'-prems: knights-path (board (n-5) m) ?ps2' hd ?ps2' = (int (n-5),int
m)
    last ?ps2' = (int (n-5)-1,2)

```

```

using ps2-prems mirror2-knights-path hd-mirror2 last-mirror2 by auto

have inter:  $?b5xm' \cap \text{board } (n-5) \ m = \{\}$ 
unfolding trans-board-def board-def by auto

have union:  $\text{board } n \ m = ?b5xm' \cup \text{board } (n-5) \ m$ 
using  $\langle n \geq 10 \wedge \text{even } n \rangle$  board-concatT[of n-5 m 5] by auto

have vs: valid-step (last  $?ps_1'$ ) (hd  $?ps_2'$ ) valid-step (last  $?ps_2'$ ) (hd  $?ps_1'$ )
using ps1'-prems ps2'-prems unfolding valid-step-def by auto

let  $?ps = ?ps_1' @ ?ps_2'$ 
have last  $?ps = \text{last } ?ps_2'$  hd  $?ps = \text{hd } ?ps_1'$ 
using ps1'-prems ps2'-prems knights-path-non-nil by auto
then have vs-c: valid-step (last  $?ps$ ) (hd  $?ps$ )
using vs by auto

have knights-path (board  $n \ m$ )  $?ps$ 
using ps1'-prems ps2'-prems inter union vs knights-path-append by auto
then show ?thesis
using vs-c unfolding knights-circuit-def by blast
qed

```

Final Theorem 1: For every $n \times m$ -board with $\min n \ m \geq 5$ and $n*m$ even there exists a Knight's circuit.

```

theorem knights-circuit-exists:
assumes  $\min n \ m \geq 5 \ \text{even } (n*m)$ 
shows  $\exists ps. \text{knights-circuit } (\text{board } n \ m) \ ps$ 
using assms
proof –
have  $n = 6 \vee m = 6 \vee n = 8 \vee m = 8 \vee (n \geq 10 \wedge \text{even } n) \vee (m \geq 10 \wedge \text{even } m)$ 
using assms by auto
then show ?thesis
proof (elim disjE)
assume  $n = 6$ 
then show ?thesis
using assms knights-path-6xm-exists by auto
next
assume  $m = 6$ 
then obtain ps where knights-circuit (board  $m \ n$ ) ps
using assms knights-path-6xm-exists by auto
then show ?thesis
using transpose-knights-circuit by auto
next
assume  $n = 8$ 
then show ?thesis
using assms knights-path-8xm-exists by auto
next

```

```

    assume  $m = 8$ 
    then obtain  $ps$  where  $knights-circuit$  (board  $m$   $n$ )  $ps$ 
      using  $assms$   $knights-path-8xm-exists$  by  $auto$ 
    then show  $?thesis$ 
      using  $transpose-knights-circuit$  by  $auto$ 
  next
    assume  $n \geq 10 \wedge even\ n$ 
    then show  $?thesis$ 
      using  $assms$   $knights-circuit-exists-n-even-gr-10$  by  $auto$ 
  next
    assume  $m \geq 10 \wedge even\ m$ 
    then obtain  $ps$  where  $knights-circuit$  (board  $m$   $n$ )  $ps$ 
      using  $assms$   $knights-circuit-exists-n-even-gr-10$  by  $auto$ 
    then show  $?thesis$ 
      using  $transpose-knights-circuit$  by  $auto$ 
  qed
qed

```

Final Theorem 2: for every $n \times m$ -board with $\min n\ m \geq 5$ there exists a Knight's path.

```

theorem  $knights-path-exists$ :
  assumes  $\min n\ m \geq 5$ 
  shows  $\exists ps. knights-path$  (board  $n$   $m$ )  $ps$ 
  using  $assms$ 
proof -
  have  $odd\ n \vee odd\ m \vee even\ (n*m)$  by  $simp$ 
  then show  $?thesis$ 
  proof (elim  $disjE$ )
    assume  $odd\ n$ 
    then show  $?thesis$ 
      using  $assms$   $knights-path-odd-n-exists$  by  $auto$ 
  next
    assume  $odd\ m$ 
    then obtain  $ps$  where  $knights-path$  (board  $m$   $n$ )  $ps$ 
      using  $assms$   $knights-path-odd-n-exists$  by  $auto$ 
    then show  $?thesis$ 
      using  $transpose-knights-path$  by  $auto$ 
  next
    assume  $even\ (n*m)$ 
    then show  $?thesis$ 
      using  $assms$   $knights-circuit-exists$  by (auto simp:  $knights-circuit-def$ )
  qed
qed
THE END
end

```