Knight's Tour Revisited Revisited

Lukas Koller
Department of Informatics
Technical University of Munich

March 17, 2025

Abstract

This is a formalization of the article "Knight's Tour Revisited" by Cull and De Curtins where they prove the existence of a Knight's path for arbitrary $n \times m$-boards with $\min(n, m) \geq 5$. If $n \cdot m$ is even, then there exists a Knight's circuit.

A Knight's Path is a sequence of moves of a Knight on a chessboard s.t. the Knight visits every square of a chessboard exactly once. Finding a Knight's path is a an instance of the Hamiltonian path problem.

During the formalization two mistakes in the original proof were discovered. These mistakes are corrected in this formalization.

Contents

theory *KnightsTour*
  imports *Main*
begin

1 Introduction and Definitions

A Knight's path is a sequence of moves on a chessboard s.t. every step in sequence is a valid move for a Knight and that the Knight visits every square on the boards exactly once. 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. Finding a Knight's path is an instance of the Hamiltonian Path Problem. 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.

Cull and De Curtins [1] prove the existence of a Knight's path on a $n\times m$-board for sufficiently large n and m. The main idea for the proof is to inductivly construct a Knight's path for the $n\times m$-board from a few pre-computed Knight's paths for small boards, i.e. 5×5, 8×6, ..., 8×9. The paths for small boards are transformed (i.e. transpose, mirror, translate) and concatenated to create paths for larger boards.

While formalizing the proofs I discovered two mistakes in the original proof in [1]: (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 incorrect: 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. In this formalization I have replaced the two incorrect paths with correct paths.

A square on a board is identified by its coordinates.

type-synonym *square* = *int* × *int*

A board is represented as a set of squares. Note, that this allows boards to have an arbitrary shape and do not necessarily need to be rectangular.

type-synonym *board* = *square set*

A (rectangular) $(n \times m)$-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, and (n,m) is the upper-right corner.

definition *board* :: *nat* ⇒ *nat* ⇒ *board* **where**
$board\ n\ m = \{(i,j)\ |i\ j.\ 1 \leq i \wedge i \leq int\ n \wedge 1 \leq j \wedge j \leq int\ m\}$

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 *path* = *square list*

A Knight can only move two squares vertically and one square horizontally or two squares horizontally and one square vertically. Thus, a knight at position (i,j) can only move to $(i\pm1,j\pm2)$ or $(i\pm2,j\pm1)$.

definition *valid-step* :: *square* ⇒ *square* ⇒ *bool* **where**
$valid\text{-}step\ s_i\ s_j \equiv (case\ s_i\ of\ (i,j) \Rightarrow s_j \in \{(i+1,j+2),(i-1,j+2),(i+1,j-2),(i-1,j-2),$
$(i+2,j+1),(i-2,j+1),(i+2,j-1),(i-2,j-1)\})$

Now we define an inductive predicate that characterizes a Knight's path. A square s_i can be pre-pended to a current Knight's path $s_j\#ps$ if (i) there is a valid step from the square s_i to the first square s_j of the current path and (ii) the square s_i has not been visited yet.

inductive *knights-path* :: *board* ⇒ *path* ⇒ *bool* **where**
$knights\text{-}path\ \{s_i\}\ [s_i]$
$|\ s_i \notin b \Longrightarrow valid\text{-}step\ s_i\ s_j \Longrightarrow knights\text{-}path\ b\ (s_j\#ps) \Longrightarrow knights\text{-}path\ (b \cup \{s_i\})$
$(s_i\#s_j\#ps)$

code-pred *knights-path* ⟨*proof*⟩

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

definition *knights-circuit b ps* ≡ (*knights-path b ps* ∧ *valid-step* (*last ps*) (*hd ps*))

2 Executable Checker for a Knight's Path

This section gives the implementation and correctness-proof for an executable checker for a knights-path w.r.t. the definition *knights-path*.

2.1 Implementation of an Executable Checker

fun *row-exec* :: *nat* ⇒ *int set* **where**
  row-exec 0 = {}
| *row-exec m* = *insert* (*int m*) (*row-exec* (*m*−*1*))

fun *board-exec-aux* :: *nat* ⇒ *int set* ⇒ *board* **where**
  board-exec-aux 0 M = {}
| *board-exec-aux k M* = {(*int k*,*j*) |*j*. *j* ∈ *M*} ∪ *board-exec-aux* (*k*−*1*) *M*

Compute a board.

fun *board-exec* :: *nat* ⇒ *nat* ⇒ *board* **where**
  board-exec n m = *board-exec-aux n* (*row-exec m*)

fun *step-checker* :: *square* ⇒ *square* ⇒ *bool* **where**
  step-checker (*i*,*j*) (*i'*,*j'*) =
    ((*i*+*1*,*j*+*2*) = (*i'*,*j'*) ∨ (*i*−*1*,*j*+*2*) = (*i'*,*j'*) ∨ (*i*+*1*,*j*−*2*) = (*i'*,*j'*) ∨ (*i*−*1*,*j*−*2*) = (*i'*,*j'*)
    ∨ (*i*+*2*,*j*+*1*) = (*i'*,*j'*) ∨ (*i*−*2*,*j*+*1*) = (*i'*,*j'*) ∨ (*i*+*2*,*j*−*1*) = (*i'*,*j'*) ∨ (*i*−*2*,*j*−*1*) = (*i'*,*j'*))

fun *path-checker* :: *board* ⇒ *path* ⇒ *bool* **where**
  path-checker b [] = *False*
| *path-checker b* [s_i] = ({s_i} = *b*)
| *path-checker b* (s_i#s_j#*ps*) = (s_i ∈ *b* ∧ *step-checker* s_i s_j ∧ *path-checker* (*b* − {s_i}) (s_j#*ps*))

fun *circuit-checker* :: *board* ⇒ *path* ⇒ *bool* **where**
  circuit-checker b ps = (*path-checker b ps* ∧ *step-checker* (*last ps*) (*hd ps*))

2.2 Correctness Proof of the Executable Checker

lemma *row-exec-leq*: *j* ∈ *row-exec m* ⟷ *1* ≤ *j* ∧ *j* ≤ *int m*
  ⟨*proof*⟩

lemma *board-exec-aux-leq-mem*: (*i*,*j*) ∈ *board-exec-aux k M* ⟷ *1* ≤ *i* ∧ *i* ≤ *int k* ∧ *j* ∈ *M*
  ⟨*proof*⟩

lemma *board-exec-leq*: (*i*,*j*) ∈ *board-exec n m* ⟷ *1* ≤ *i* ∧ *i* ≤ *int n* ∧ *1* ≤ *j* ∧ *j* ≤ *int m*
  ⟨*proof*⟩

lemma *board-exec-correct*: *board n m* = *board-exec n m*
  ⟨*proof*⟩

lemma *step-checker-correct*: *step-checker* s_i s_j ⟷ *valid-step* s_i s_j
⟨*proof*⟩

lemma *step-checker-rev*: *step-checker* (*i*,*j*) (*i'*,*j'*) ⟹ *step-checker* (*i'*,*j'*) (*i*,*j*)

⟨*proof*⟩

lemma *knights-path-intro-rev*:
assumes $s_i \in b$ *valid-step* s_i s_j *knights-path* $(b - \{s_i\})$ $(s_j\#ps)$
shows *knights-path* b $(s_i\#s_j\#ps)$
⟨*proof*⟩

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

lemma *path-checker-correct*: *path-checker* b $ps \longleftrightarrow$ *knights-path* b ps
⟨*proof*⟩

corollary *knights-path-exec-simp*: *knights-path* (*board* n m) $ps \longleftrightarrow$ *path-checker* (*board-exec* n m) ps
⟨*proof*⟩

lemma *circuit-checker-correct*: *circuit-checker* b $ps \longleftrightarrow$ *knights-circuit* b ps
⟨*proof*⟩

corollary *knights-circuit-exec-simp*:
knights-circuit (*board* n m) $ps \longleftrightarrow$ *circuit-checker* (*board-exec* n m) ps
⟨*proof*⟩

3 Basic Properties of *knights-path* and *knights-circuit*

lemma *board-leq-subset*: $n_1 \leq n_2 \wedge m_1 \leq m_2 \Longrightarrow$ *board* n_1 $m_1 \subseteq$ *board* n_2 m_2
⟨*proof*⟩

lemma *finite-row-exec*: *finite* (*row-exec* m)
⟨*proof*⟩

lemma *finite-board-exec-aux*: *finite* $M \Longrightarrow$ *finite* (*board-exec-aux* n M)
⟨*proof*⟩

lemma *board-finite*: *finite* (*board* n m)
⟨*proof*⟩

lemma *card-row-exec*: *card* (*row-exec* m) = m
⟨*proof*⟩

lemma *set-comp-ins*:
$\{(k,j) \mid j.\ j \in$ *insert* x $M\}$ = *insert* (k,x) $\{(k,j) \mid j.\ j \in M\}$ (**is** *?Mi* = *?iM*)
⟨*proof*⟩

lemma *finite-card-set-comp*: *finite* $M \Longrightarrow$ *card* $\{(k,j) \mid j.\ j \in M\}$ = *card* M
⟨*proof*⟩

lemma *card-board-exec-aux*: *finite* $M \Longrightarrow$ *card* (*board-exec-aux* k M) = $k *$ *card* M
⟨*proof*⟩

lemma *card-board*: *card* (*board n m*) = *n* ∗ *m*
⟨*proof*⟩

lemma *knights-path-board-non-empty*: *knights-path b ps* ⟹ *b* ≠ {}
⟨*proof*⟩

lemma *knights-path-board-m-n-geq-1*: *knights-path* (*board n m*) *ps* ⟹ *min n m* ≥ *1*
⟨*proof*⟩

lemma *knights-path-non-nil*: *knights-path b ps* ⟹ *ps* ≠ []
⟨*proof*⟩

lemma *knights-path-set-eq*: *knights-path b ps* ⟹ *set ps* = *b*
⟨*proof*⟩

lemma *knights-path-subset*:
knights-path b_1 ps_1 ⟹ *knights-path* b_2 ps_2 ⟹ *set* ps_1 ⊆ *set* ps_2 ⟷ $b_1 \subseteq b_2$
⟨*proof*⟩

lemma *knights-path-board-unique*: *knights-path* b_1 *ps* ⟹ *knights-path* b_2 *ps* ⟹ $b_1 = b_2$
⟨*proof*⟩

lemma *valid-step-neq*: *valid-step* s_i s_j ⟹ $s_i \neq s_j$
⟨*proof*⟩

lemma *valid-step-non-transitive*: *valid-step* s_i s_j ⟹ *valid-step* s_j s_k ⟹ ¬*valid-step* s_i s_k
⟨*proof*⟩

lemma *knights-path-distinct*: *knights-path b ps* ⟹ *distinct ps*
⟨*proof*⟩

lemma *knights-path-length*: *knights-path b ps* ⟹ *length ps* = *card b*
⟨*proof*⟩

lemma *knights-path-take*:
assumes *knights-path b ps* *0* < *k* *k* < *length ps*
shows *knights-path* (*set* (*take k ps*)) (*take k ps*)
⟨*proof*⟩

lemma *knights-path-drop*:
assumes *knights-path b ps* *0* < *k* *k* < *length ps*
shows *knights-path* (*set* (*drop k ps*)) (*drop k ps*)
⟨*proof*⟩

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
$\exists b_1\ b_2.$ *knights-path* b_1 (*take k ps*) ∧ *knights-path* b_2 (*drop k ps*) ∧ $b_1 \cup b_2 = b$ ∧ $b_1 \cap b_2 = \{\}$
⟨*proof*⟩

Append two disjoint Knight's paths.

corollary *knights-path-append*:
assumes *knights-path* b_1 ps_1 *knights-path* b_2 ps_2 $b_1 \cap b_2 = \{\}$ *valid-step* (*last* ps_1) (*hd* ps_2)
shows *knights-path* ($b_1 \cup b_2$) (ps_1 @ ps_2)
⟨*proof*⟩

lemma *valid-step-rev*: *valid-step* s_i s_j ⟹ *valid-step* s_j s_i
⟨*proof*⟩

Reverse a Knight's path.

corollary *knights-path-rev*:
assumes *knights-path b ps*
shows *knights-path b* (*rev ps*)
⟨*proof*⟩

Reverse a Knight's circuit.

corollary *knights-circuit-rev*:
assumes *knights-circuit b ps*
shows *knights-circuit b* (*rev ps*)
⟨*proof*⟩

lemma *knights-circuit-rotate1*:
assumes *knights-circuit b* (s_i#*ps*)
shows *knights-circuit b* (*ps*@[s_i])
⟨*proof*⟩

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

lemma *knights-circuit-rotate-to*:
assumes *knights-circuit b ps hd* (*drop k ps*) = s_i *k* < *length ps*
shows $\exists ps'.$ *knights-circuit b ps'* ∧ *hd ps'* = s_i
⟨*proof*⟩

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*) ∨ (*i*,*j*) = (*3*,*2*)
⟨*proof*⟩

lemma *list-len-g-1-split*: $length\ xs > 1 \Longrightarrow \exists x_1\ x_2\ xs'.\ xs = x_1\#x_2\#xs'$
⟨*proof*⟩

lemma *list-len-g-3-split*: $length\ xs > 3 \Longrightarrow \exists x_1\ x_2\ xs'\ x_3.\ xs = x_1\#x_2\#xs'@[x_3]$
⟨*proof*⟩

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 min n m* $\geq$ *5*
shows $\exists$*ps. knights-circuit (board n m) ps* $\wedge$ *hd ps = (1,1)* $\wedge$ *last ps = (3,2)*
⟨*proof*⟩

4 Transposing Paths and Boards

4.1 Implementation of Path and Board Transposition

definition *transpose-square* s_i = (*case* s_i *of* (*i*,*j*) ⇒ (*j*,*i*))

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

definition *transpose-board* :: *board* ⇒ *board* **where**
transpose-board b ≡ {(*j*,*i*) |*i j*. (*i*,*j*) ∈ *b*}

4.2 Correctness of Path and Board Transposition

lemma *transpose2*: *transpose-square* (*transpose-square* s_i) = s_i
⟨*proof*⟩

lemma *transpose-nil*: *ps* = [] ⟷ *transpose ps* = []
⟨*proof*⟩

lemma *transpose-length*: *length ps* = *length* (*transpose ps*)
⟨*proof*⟩

lemma *hd-transpose*: *ps* ≠[] ⟹ *hd* (*transpose ps*) = *transpose-square* (*hd ps*)
⟨*proof*⟩

lemma *last-transpose*: *ps* ≠[] ⟹ *last* (*transpose ps*) = *transpose-square* (*last ps*)
⟨*proof*⟩

lemma *take-transpose*:
shows *take k* (*transpose ps*) = *transpose* (*take k ps*)
⟨*proof*⟩

lemma *drop-transpose*:
shows *drop k* (*transpose ps*) = *transpose* (*drop k ps*)

⟨*proof*⟩

lemma *transpose-board-correct*: $s_i \in b \longleftrightarrow$ (*transpose-square* s_i) $\in$ *transpose-board* *b*
⟨*proof*⟩

lemma *transpose-board*: *transpose-board* (*board n m*) = *board m n*
⟨*proof*⟩

lemma *insert-transpose-board*:
insert (*transpose-square* s_i) (*transpose-board b*) = *transpose-board* (*insert* s_i *b*)
⟨*proof*⟩

lemma *transpose-board2*: *transpose-board* (*transpose-board b*) = *b*
⟨*proof*⟩

lemma *transpose-union*: *transpose-board* ($b_1 \cup b_2$) = *transpose-board* b_1 $\cup$ *transpose-board* b_2
⟨*proof*⟩

lemma *transpose-valid-step*:
valid-step s_i s_j $\longleftrightarrow$ *valid-step* (*transpose-square* s_i) (*transpose-square* s_j)
⟨*proof*⟩

lemma *transpose-knights-path'*:
assumes *knights-path b ps*
shows *knights-path* (*transpose-board b*) (*transpose ps*)
⟨*proof*⟩

corollary *transpose-knights-path*:
assumes *knights-path* (*board n m*) *ps*
shows *knights-path* (*board m n*) (*transpose ps*)
⟨*proof*⟩

corollary *transpose-knights-circuit*:
assumes *knights-circuit* (*board n m*) *ps*
shows *knights-circuit* (*board m n*) (*transpose ps*)
⟨*proof*⟩

5 Mirroring Paths and Boards

5.1 Implementation of Path and Board Mirroring

abbreviation *min1 ps* ≡ *Min* ((*fst*) ‘ *set ps*)
abbreviation *max1 ps* ≡ *Max* ((*fst*) ‘ *set ps*)
abbreviation *min2 ps* ≡ *Min* ((*snd*) ‘ *set ps*)
abbreviation *max2 ps* ≡ *Max* ((*snd*) ‘ *set ps*)

definition *mirror1-square* :: *int* ⇒ *square* ⇒ *square* **where**

mirror1-square n s_i = (*case* s_i *of* (*i*,*j*) ⇒ (*n*−*i*,*j*))

fun *mirror1-aux* :: *int* ⇒ *path* ⇒ *path* **where**
mirror1-aux *n* [] = []
| *mirror1-aux* *n* (s_i#*ps*) = (*mirror1-square* *n* s_i)#*mirror1-aux* *n* *ps*

definition *mirror1* *ps* = *mirror1-aux* (*max1* *ps* + *min1* *ps*) *ps*

definition *mirror1-board* :: *int* ⇒ *board* ⇒ *board* **where**
mirror1-board *n* *b* ≡ {*mirror1-square* *n* s_i |s_i. s_i ∈ *b*}

definition *mirror2-square* :: *int* ⇒ *square* ⇒ *square* **where**
mirror2-square *m* s_i = (*case* s_i *of* (*i*,*j*) ⇒ (*i*,*m*−*j*))

fun *mirror2-aux* :: *int* ⇒ *path* ⇒ *path* **where**
mirror2-aux *m* [] = []
| *mirror2-aux* *m* (s_i#*ps*) = (*mirror2-square* *m* s_i)#*mirror2-aux* *m* *ps*

definition *mirror2* *ps* = *mirror2-aux* (*max2* *ps* + *min2* *ps*) *ps*

definition *mirror2-board* :: *int* ⇒ *board* ⇒ *board* **where**
mirror2-board *m* *b* ≡ {*mirror2-square* *m* s_i |s_i. s_i ∈ *b*}

5.2 Correctness of Path and Board Mirroring

lemma *mirror1-board-id*: *mirror1-board* (*int* *n*+*1*) (*board* *n* *m*) = *board* *n* *m* (**is** - = *?b*)
⟨*proof*⟩

lemma *mirror2-board-id*: *mirror2-board* (*int* *m*+*1*) (*board* *n* *m*) = *board* *n* *m* (**is** - = *?b*)
⟨*proof*⟩

lemma *knights-path-min1*: *knights-path* (*board* *n* *m*) *ps* ⟹ *min1* *ps* = *1*
⟨*proof*⟩

lemma *knights-path-min2*: *knights-path* (*board* *n* *m*) *ps* ⟹ *min2* *ps* = *1*
⟨*proof*⟩

lemma *knights-path-max1*: *knights-path* (*board* *n* *m*) *ps* ⟹ *max1* *ps* = *int* *n*
⟨*proof*⟩

lemma *knights-path-max2*: *knights-path* (*board* *n* *m*) *ps* ⟹ *max2* *ps* = *int* *m*
⟨*proof*⟩

lemma *mirror1-aux-nil*: *ps* = [] ⟷ *mirror1-aux* *m* *ps* = []
⟨*proof*⟩

lemma *mirror1-nil*: *ps* = [] ⟷ *mirror1* *ps* = []

⟨*proof*⟩

lemma *mirror2-aux-nil*: $ps = [] \longleftrightarrow mirror2\text{-}aux\ m\ ps = []$
⟨*proof*⟩

lemma *mirror2-nil*: $ps = [] \longleftrightarrow mirror2\ ps = []$
⟨*proof*⟩

lemma *length-mirror1-aux*: $length\ ps = length\ (mirror1\text{-}aux\ n\ ps)$
⟨*proof*⟩

lemma *length-mirror1*: $length\ ps = length\ (mirror1\ ps)$
⟨*proof*⟩

lemma *length-mirror2-aux*: $length\ ps = length\ (mirror2\text{-}aux\ n\ ps)$
⟨*proof*⟩

lemma *length-mirror2*: $length\ ps = length\ (mirror2\ ps)$
⟨*proof*⟩

lemma *mirror1-board-iff*: $s_i \notin b \longleftrightarrow mirror1\text{-}square\ n\ s_i \notin mirror1\text{-}board\ n\ b$
⟨*proof*⟩

lemma *mirror2-board-iff*: $s_i \notin b \longleftrightarrow mirror2\text{-}square\ n\ s_i \notin mirror2\text{-}board\ n\ b$
⟨*proof*⟩

lemma *insert-mirror1-board*:
$insert\ (mirror1\text{-}square\ n\ s_i)\ (mirror1\text{-}board\ n\ b) = mirror1\text{-}board\ n\ (insert\ s_i\ b)$
⟨*proof*⟩

lemma *insert-mirror2-board*:
$insert\ (mirror2\text{-}square\ n\ s_i)\ (mirror2\text{-}board\ n\ b) = mirror2\text{-}board\ n\ (insert\ s_i\ b)$
⟨*proof*⟩

lemma *valid-step-mirror1*:
$valid\text{-}step\ s_i\ s_j \longleftrightarrow valid\text{-}step\ (mirror1\text{-}square\ n\ s_i)\ (mirror1\text{-}square\ n\ s_j)$
⟨*proof*⟩

lemma *valid-step-mirror2*:
$valid\text{-}step\ s_i\ s_j \longleftrightarrow valid\text{-}step\ (mirror2\text{-}square\ m\ s_i)\ (mirror2\text{-}square\ m\ s_j)$
⟨*proof*⟩

lemma *hd-mirror1*:
assumes $knights\text{-}path\ (board\ n\ m)\ ps\ hd\ ps = (i,j)$
shows $hd\ (mirror1\ ps) = (int\ n+1-i,j)$
⟨*proof*⟩

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

shows *last* (*mirror1-aux n ps*) = (*n*−*i*,*j*)
⟨*proof*⟩

lemma *last-mirror1*:
assumes *knights-path* (*board n m*) *ps last ps* = (*i*,*j*)
shows *last* (*mirror1 ps*) = (*int n*+*1*−*i*,*j*)
⟨*proof*⟩

lemma *hd-mirror2*:
assumes *knights-path* (*board n m*) *ps hd ps* = (*i*,*j*)
shows *hd* (*mirror2 ps*) = (*i*,*int m*+*1*−*j*)
⟨*proof*⟩

lemma *last-mirror2-aux*:
assumes *ps* ≠ [] *last ps* = (*i*,*j*)
shows *last* (*mirror2-aux m ps*) = (*i*,*m*−*j*)
⟨*proof*⟩

lemma *last-mirror2*:
assumes *knights-path* (*board n m*) *ps last ps* = (*i*,*j*)
shows *last* (*mirror2 ps*) = (*i*,*int m*+*1*−*j*)
⟨*proof*⟩

lemma *mirror1-aux-knights-path*:
assumes *knights-path b ps*
shows *knights-path* (*mirror1-board n b*) (*mirror1-aux n ps*)
⟨*proof*⟩

corollary *mirror1-knights-path*:
assumes *knights-path* (*board n m*) *ps*
shows *knights-path* (*board n m*) (*mirror1 ps*)
⟨*proof*⟩

lemma *mirror2-aux-knights-path*:
assumes *knights-path b ps*
shows *knights-path* (*mirror2-board n b*) (*mirror2-aux n ps*)
⟨*proof*⟩

corollary *mirror2-knights-path*:
assumes *knights-path* (*board n m*) *ps*
shows *knights-path* (*board n m*) (*mirror2 ps*)
⟨*proof*⟩

5.3 Rotate Knight's Paths

Transposing (*KnightsTour.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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

6 Translating Paths and Boards

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.

6.1 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 (k_1,k_2) [] = []
| *trans-path* (k_1,k_2) $((i,j)\#xs) = (i+k_1,j+k_2)\#($*trans-path* (k_1,k_2) *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* $(k_1,k_2) \Rightarrow \{(i+k_1,j+k_2)|i\ j.\ (i,j) \in b\}$)

6.2 Correctness of Path and Board Translation

lemma *trans-path-length*: *length ps* = *length* (*trans-path* (k_1,k_2) *ps*)
⟨*proof*⟩

lemma *trans-path-non-nil*: *ps* ≠ [] ⟹ *trans-path* (k_1,k_2) *ps* ≠ []
⟨*proof*⟩

lemma *trans-path-correct*: $(i,j) \in$ *set ps* ⟷ $(i+k_1,j+k_2) \in$ *set* (*trans-path* (k_1,k_2) *ps*)
⟨*proof*⟩

lemma *trans-path-non-nil-last*:
ps ≠ [] ⟹ *last* (*trans-path* (k_1,k_2) *ps*) = *last* (*trans-path* (k_1,k_2) $((i,j)\#ps)$)
⟨*proof*⟩

lemma *hd-trans-path*:
  assumes *ps* $\neq$ [] *hd ps* = (*i*,*j*)
  shows *hd* (*trans-path* (k_1,k_2) *ps*) = (*i*+k_1,*j*+k_2)
  ⟨*proof*⟩

lemma *last-trans-path*:
  assumes *ps* $\neq$ [] *last ps* = (*i*,*j*)
  shows *last* (*trans-path* (k_1,k_2) *ps*) = (*i*+k_1,*j*+k_2)
  ⟨*proof*⟩

lemma *take-trans*:
  shows *take k* (*trans-path* (k_1,k_2) *ps*) = *trans-path* (k_1,k_2) (*take k ps*)
⟨*proof*⟩

lemma *drop-trans*:
  shows *drop k* (*trans-path* (k_1,k_2) *ps*) = *trans-path* (k_1,k_2) (*drop k ps*)
⟨*proof*⟩

lemma *trans-board-correct*: (*i*,*j*) $\in$ *b* $\longleftrightarrow$ (*i*+k_1,*j*+k_2) $\in$ *trans-board* (k_1,k_2) *b*
  ⟨*proof*⟩

lemma *board-subset*: $n_1 \leq n_2 \Longrightarrow m_1 \leq m_2 \Longrightarrow$ *board* n_1 m_1 $\subseteq$ *board* n_2 m_2
  ⟨*proof*⟩

Board concatenation

corollary *board-concat*:
  shows *board n* m_1 $\cup$ *trans-board* (*0*,*int* m_1) (*board n* m_2) = *board n* (m_1+m_2)
(**is** *?b1* $\cup$ *?b2* = *?b*)
⟨*proof*⟩

lemma *transpose-trans-board*:
  transpose-board (*trans-board* (k_1,k_2) *b*) = *trans-board* (k_2,k_1) (*transpose-board b*)
  ⟨*proof*⟩

corollary *board-concatT*:
  shows *board* n_1 *m* $\cup$ *trans-board* (*int* n_1,*0*) (*board* n_2 *m*) = *board* (n_1+n_2) *m* (**is**
?b_1 $\cup$ *?*b_2 = *?b*)
⟨*proof*⟩

lemma *trans-valid-step*:
  valid-step (*i*,*j*) (i',j') $\Longrightarrow$ *valid-step* (*i*+k_1,*j*+k_2) (i'+k_1,j'+k_2)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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

definition *step-in* :: *path* ⇒ *square* ⇒ *square* ⇒ *bool* **where**
step-in ps s_i s_j ≡ (∃ *k*. *0* < *k* ∧ *k* < *length ps* ∧ *last* (*take k ps*) = s_i ∧ *hd* (*drop k ps*) = s_j)

lemma *step-in-Cons*: *step-in ps* s_i s_j ⟹ *step-in* (s_k#*ps*) s_i s_j
⟨*proof*⟩

lemma *step-in-append*: *step-in ps* s_i s_j ⟹ *step-in* (*ps*@*ps'*) s_i s_j
⟨*proof*⟩

lemma *step-in-prepend*: *step-in ps* s_i s_j ⟹ *step-in* (*ps'*@*ps*) s_i s_j
⟨*proof*⟩

lemma *step-in-valid-step*: *knights-path b ps* ⟹ *step-in ps* s_i s_j ⟹ *valid-step* s_i s_j
⟨*proof*⟩

lemma *trans-step-in*:
step-in ps (*i*,*j*) (*i'*,*j'*) ⟹ *step-in* (*trans-path* (k_1,k_2) *ps*) ($i+k_1$,$j+k_2$) ($i'+k_1$,$j'+k_2$)
⟨*proof*⟩

lemma *transpose-step-in*:
step-in ps s_i s_j ⟹ *step-in* (*transpose ps*) (*transpose-square* s_i) (*transpose-square* s_j)
(**is** - ⟹ *step-in ?psT ?*s_i*T ?*s_j*T*)
⟨*proof*⟩

lemma *hd-take*: *0* < *k* ⟹ *hd xs* = *hd* (*take k xs*)
⟨*proof*⟩

lemma *last-drop*: *k* < *length xs* ⟹ *last xs* = *last* (*drop k xs*)
⟨*proof*⟩

6.3 Concatenate Knight's Paths and Circuits

Concatenate two knight's path on a $n \times m$-board along the 2nd axis if the first path contains the step $s_i \to s_j$ and there are valid steps $s_i \to$ *hd* ps_2' and $s_j \to$ *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* 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
shows ∃ *ps*. *knights-path* (*board n* (m_1+m_2)) *ps* ∧ *hd ps* = *hd* ps_1
∧ *last ps* = *last* ps_1 ∧ *step-in ps* (*i*,*int* m_1+j) (*i'*,*int* m_1+j')
⟨*proof*⟩

lemma *len1-hd-last*: $length\ xs = 1 \Longrightarrow hd\ xs = last\ xs$
⟨*proof*⟩

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

corollary *knights-path-split-concat*:
assumes $knights\text{-}path\ (board\ n\ m_1)\ ps_1\ knights\text{-}path\ (board\ n\ m_2)\ ps_2$
$step\text{-}in\ ps_1\ s_i\ s_j\ hd\ ps_2 = (i_h,j_h)\ last\ ps_2 = (i_l,j_l)$
$valid\text{-}step\ s_i\ (i_h,int\ m_1+j_h)\ valid\text{-}step\ (i_l,int\ m_1+j_l)\ s_j$
shows $\exists ps.\ knights\text{-}path\ (board\ n\ (m_1+m_2))\ ps \wedge hd\ ps = hd\ ps_1 \wedge last\ ps = last\ ps_1$
⟨*proof*⟩

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

corollary *knights-path-split-concatT*:
assumes $knights\text{-}path\ (board\ n_1\ m)\ ps_1\ knights\text{-}path\ (board\ n_2\ m)\ ps_2$
$step\text{-}in\ ps_1\ s_i\ s_j\ hd\ ps_2 = (i_h,j_h)\ last\ ps_2 = (i_l,j_l)$
$valid\text{-}step\ s_i\ (int\ n_1+i_h,j_h)\ valid\text{-}step\ (int\ n_1+i_l,j_l)\ s_j$
shows $\exists ps.\ knights\text{-}path\ (board\ (n_1+n_2)\ m)\ ps \wedge hd\ ps = hd\ ps_1 \wedge last\ ps = last\ ps_1$
⟨*proof*⟩

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*⟩

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 $knights\text{-}path\ (board\ n\ m_1)\ ps_1\ knights\text{-}path\ (board\ n\ m_2)\ ps_2$
$last\ ps_1 = (2,int\ m_1-1)\ hd\ ps_2 = (1,1)$
shows $knights\text{-}path\ (board\ n\ (m_1+m_2))\ (ps_1\ @\ (trans\text{-}path\ (0,int\ m_1)\ ps_2))$
⟨*proof*⟩

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 cirucit starts in $(1,1)$ (lower-left) and end in $(3,2)$.

corollary *knights-circuit-lr-concat*:
assumes *knights-circuit* (*board n* m_1) ps_1 *knights-circuit* (*board n* m_2) ps_2
step-in ps_1 (*2*,*int* m_1−*1*) (*4*,*int* m_1)
hd ps_2 = (*1*,*1*) *last* ps_2 = (*3*,*2*) *step-in* ps_2 (*2*,*int* m_2−*1*) (*4*,*int* m_2)
shows ∃ *ps*. *knights-circuit* (*board n* (m_1+m_2)) *ps* ∧ *step-in ps* (*2*,*int* (m_1+m_2)−*1*) (*4*,*int* (m_1+m_2))
⟨*proof*⟩

7 Parsing Paths

In this section functions are implemented to parse and construct paths. The parser converts the matrix representation ((*nat list*) *list*) used in [1] to a path (*path*).

for debugging

```
fun test-path :: path ⇒ bool where
  test-path (s_i#s_j#xs) = (step-checker s_i s_j ∧ test-path (s_j#xs))
| test-path - = True

fun f-opt :: ('a ⇒ 'a) ⇒ 'a option ⇒ 'a option where
  f-opt - None = None
| f-opt f (Some a) = Some (f a)

fun add-opt-fst-sq :: int ⇒ square option ⇒ 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 ⇒ nat list ⇒ 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 ⇒ (nat list) list ⇒ square option where
  find-k-sqr k [] = None
| find-k-sqr k (r#rs) = (case find-k-in-col k r of
      None ⇒ f-opt (λ(i,j). (i+1,j)) (find-k-sqr k rs)
    | Some j ⇒ Some (1,j))
```

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

```
fun to-sqrs :: nat ⇒ (nat list) list ⇒ path option where
  to-sqrs 0 rs = Some []
| to-sqrs k rs = (case find-k-sqr k rs of
      None ⇒ None
    | Some s_i ⇒ f-opt (λps. ps@[s_i]) (to-sqrs (k−1) rs))

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

Convert a matrix (*nat list list*) to a path (*path*). With this function we implicitly define the lower-left corner to be (*1*,*1*) and the upper-right corner to be (*n*,*m*).

definition *to-path rs* ≡ *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*]]

8 Knight's 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 show the visited squares numbered in ascending order.

abbreviation *b5x5* ≡ *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* ≡ *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-5x5-lr*: *knights-path b5x5 kp5x5lr*
  ⟨*proof*⟩

lemma *kp-5x5-lr-hd*: *hd kp5x5lr* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x5-lr-last*: *last kp5x5lr* = (*2*,*4*) ⟨*proof*⟩

lemma *kp-5x5-lr-non-nil*: *kp5x5lr* ≠ [] ⟨*proof*⟩

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* ≡ *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*
  ⟨*proof*⟩

lemma *kp-5x5-ur-hd*: *hd kp5x5ur* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x5-ur-last*: *last kp5x5ur* = (*4*,*4*) ⟨*proof*⟩

lemma *kp-5x5-ur-non-nil*: *kp5x5ur* ≠ [] ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *kp-5x6-lr-hd*: *hd kp5x6lr* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x6-lr-last*: *last kp5x6lr* = (*2*,*5*) ⟨*proof*⟩

lemma *kp-5x6-lr-non-nil*: *kp5x6lr* ≠ [] ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *kp-5x6-ur-hd*: *hd kp5x6ur* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x6-ur-last*: *last kp5x6ur* = (*4*,*5*) ⟨*proof*⟩

lemma *kp-5x6-ur-non-nil*: *kp5x6ur* ≠ [] ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *kp-5x7-lr-hd*: *hd kp5x7lr* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x7-lr-last*: *last kp5x7lr* = (*2*,*6*) ⟨*proof*⟩

lemma *kp-5x7-lr-non-nil*: *kp5x7lr* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kp-5x7-ur-hd*: *hd kp5x7ur* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x7-ur-last*: *last kp5x7ur* = (*4*,*6*) ⟨*proof*⟩

lemma *kp-5x7-ur-non-nil*: *kp5x7ur* ≠ [] ⟨*proof*⟩

abbreviation *b5x8* ≡ *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* ≡ *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*
⟨*proof*⟩

lemma *kp-5x8-lr-hd*: *hd kp5x8lr* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x8-lr-last*: *last kp5x8lr* = (*2*,*7*) ⟨*proof*⟩

lemma *kp-5x8-lr-non-nil*: *kp5x8lr* ≠ [] ⟨*proof*⟩

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* ≡ *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*: *knights-path b5x8 kp5x8ur*
⟨*proof*⟩

lemma *kp-5x8-ur-hd*: *hd kp5x8ur* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x8-ur-last*: *last kp5x8ur* = (*4*,*7*) ⟨*proof*⟩

lemma *kp-5x8-ur-non-nil*: *kp5x8ur* ≠ [] ⟨*proof*⟩

abbreviation *b5x9* ≡ *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* ≡ *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*: *knights-path b5x9 kp5x9lr*
⟨*proof*⟩

lemma *kp-5x9-lr-hd*: *hd kp5x9lr* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x9-lr-last*: *last kp5x9lr* = (*2*,*8*) ⟨*proof*⟩

lemma *kp-5x9-lr-non-nil*: *kp5x9lr* ≠ [] ⟨*proof*⟩

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* ≡ *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*: *knights-path b5x9 kp5x9ur*
⟨*proof*⟩

lemma *kp-5x9-ur-hd*: *hd kp5x9ur* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-5x9-ur-last*: *last kp5x9ur* = (*4*,*8*) ⟨*proof*⟩

lemma *kp-5x9-ur-non-nil*: *kp5x9ur* ≠ [] ⟨*proof*⟩

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 *knights-path-5xm-lr-exists*:
assumes $m \geq 5$
shows $\exists ps.\ knights\text{-}path\ (board\ 5\ m)\ ps \wedge hd\ ps = (1,1) \wedge last\ ps = (2,int\ m-1)$
⟨*proof*⟩

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 $\exists ps.\ knights\text{-}path\ (board\ 5\ m)\ ps \wedge hd\ ps = (1,1) \wedge last\ ps = (4,int\ m-1)$
⟨*proof*⟩

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

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

9 Knight's 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 $kp6x5ul \equiv the\ (to\text{-}path$
$[[10,19,4,29,12],$
$[3,30,11,20,5],$
$[18,9,24,13,28],$
$[25,2,17,6,21],$
$[16,23,8,27,14],$
$[1,26,15,22,7]])$
lemma *kp-6x5-ul*: *knights-path b6x5 kp6x5ul*
⟨*proof*⟩

lemma *kp-6x5-ul-hd*: $hd\ kp6x5ul = (1,1)$ ⟨*proof*⟩

lemma *kp-6x5-ul-last*: *last kp6x5ul* = (*5*,*2*) ⟨*proof*⟩

lemma *kp-6x5-ul-non-nil*: *kp6x5ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*: *knights-circuit b6x5 kc6x5*
⟨*proof*⟩

lemma *kc-6x5-hd*: *hd kc6x5* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-6x5-non-nil*: *kc6x5* ≠ [] ⟨*proof*⟩

abbreviation *b6x6* ≡ *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-false* ≡ *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*]])

lemma ¬*knights-path b6x6 kp6x6ul-false*
⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kp-6x6-ul-hd*: *hd kp6x6ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-6x6-ul-last*: *last kp6x6ul* = (*5*,*2*) ⟨*proof*⟩

lemma *kp-6x6-ul-non-nil*: *kp6x6ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kc-6x6-hd*: *hd kc6x6* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-6x6-non-nil*: *kc6x6* ≠ [] ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *kp-6x7-ul-hd*: *hd kp6x7ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-6x7-ul-last*: *last kp6x7ul* = (*5*,*2*) ⟨*proof*⟩

lemma *kp-6x7-ul-non-nil*: *kp6x7ul* ≠ [] ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *kc-6x7-hd*: *hd kc6x7* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-6x7-non-nil*: *kc6x7* ≠ [] ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *kp-6x8-ul-hd*: *hd kp6x8ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-6x8-ul-last*: *last kp6x8ul* = (*5*,*2*) ⟨*proof*⟩

lemma *kp-6x8-ul-non-nil*: *kp6x8ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kc-6x8-hd*: *hd kc6x8* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-6x8-non-nil*: *kc6x8* ≠ [] ⟨*proof*⟩

abbreviation *b6x9* ≡ *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* ≡ *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*
⟨*proof*⟩

lemma *kp-6x9-ul-hd*: *hd kp6x9ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-6x9-ul-last*: *last kp6x9ul* = (*5*,*2*) ⟨*proof*⟩

lemma *kp-6x9-ul-non-nil*: *kp6x9ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kc-6x9-hd*: *hd kc6x9* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-6x9-non-nil*: *kc6x9* ≠ [] ⟨*proof*⟩

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 ∃*ps*. *knights-path* (*board 6 m*) *ps* ∧ *hd ps* = (*1*,*1*) ∧ *last ps* = (*5*,*2*)
⟨*proof*⟩

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

lemma *knights-circuit-6xm-exists*:
assumes $m \geq 5$
shows ∃*ps*. *knights-circuit* (*board 6 m*) *ps*
⟨*proof*⟩

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

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

10 Knight's Paths and Circuits for $8\times m$-Boards

abbreviation *b8x5* ≡ *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* ≡ *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*
⟨*proof*⟩

lemma *kp-8x5-ul-hd*: *hd kp8x5ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-8x5-ul-last*: *last kp8x5ul* = (*7*,*2*) ⟨*proof*⟩

lemma *kp-8x5-ul-non-nil*: *kp8x5ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
  ⟨*proof*⟩

lemma *kc-8x5-hd*: *hd kc8x5* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-8x5-last*: *last kc8x5* = (*3*,*2*) ⟨*proof*⟩

lemma *kc-8x5-non-nil*: *kc8x5* ≠ [] ⟨*proof*⟩

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

abbreviation *b8x6* ≡ *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* ≡ *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-8x6-ul*: *knights-path b8x6 kp8x6ul*
⟨*proof*⟩

lemma *kp-8x6-ul-hd*: *hd kp8x6ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-8x6-ul-last*: *last kp8x6ul* = (*7*,*2*) ⟨*proof*⟩

lemma *kp-8x6-ul-non-nil*: *kp8x6ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kc-8x6-hd*: *hd kc8x6* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-8x6-non-nil*: *kc8x6* ≠ [] ⟨*proof*⟩

lemma *kc-8x6-si*: *step-in kc8x6* (*2*,*5*) (*4*,*6*) (**is** *step-in ?ps - -*)
⟨*proof*⟩

abbreviation *b8x7* ≡ *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* ≡ *the* (*to-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*
⟨*proof*⟩

lemma *kp-8x7-ul-hd*: *hd kp8x7ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-8x7-ul-last*: *last kp8x7ul* = (*7*,*2*) ⟨*proof*⟩

lemma *kp-8x7-ul-non-nil*: *kp8x7ul* ≠ [] ⟨*proof*⟩

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* ≡ *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-8x7*: *knights-circuit b8x7 kc8x7*
⟨*proof*⟩

lemma *kc-8x7-hd*: *hd kc8x7* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-8x7-non-nil*: *kc8x7* ≠ [] ⟨*proof*⟩

lemma *kc-8x7-si*: *step-in kc8x7* (*2*,*6*) (*4*,*7*) (**is** *step-in ?ps* - -)
⟨*proof*⟩

abbreviation *b8x8* ≡ *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-false* ≡ *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*]])

lemma ¬*knights-path b8x8 kp8x8ul-false*
⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kp-8x8-ul-hd*: *hd kp8x8ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-8x8-ul-last*: *last kp8x8ul* = (*7*,*2*) ⟨*proof*⟩

lemma *kp-8x8-ul-non-nil*: *kp8x8ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kc-8x8-hd*: *hd kc8x8* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-8x8-non-nil*: *kc8x8* ≠ [] ⟨*proof*⟩

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

abbreviation *b8x9* ≡ *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* ≡ *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*: *knights-path b8x9 kp8x9ul*
⟨*proof*⟩

lemma *kp-8x9-ul-hd*: *hd kp8x9ul* = (*1*,*1*) ⟨*proof*⟩

lemma *kp-8x9-ul-last*: *last kp8x9ul* = (*7*,*2*) ⟨*proof*⟩

lemma *kp-8x9-ul-non-nil*: *kp8x9ul* ≠ [] ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

lemma *kc-8x9-hd*: *hd kc8x9* = (*1*,*1*) ⟨*proof*⟩

lemma *kc-8x9-non-nil*: *kc8x9* ≠ [] ⟨*proof*⟩

lemma *kc-8x9-si*: *step-in kc8x9* (*2*,*8*) (*4*,*9*) (**is** *step-in ?ps* - -)
⟨*proof*⟩

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 ∃*ps*. *knights-circuit* (*board 8 m*) *ps* ∧ *step-in ps* (*2*,*int m*−*1*) (*4*,*int m*)
⟨*proof*⟩

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 $m \geq 5$
  shows $\exists ps.\ knights\text{-}path\ (board\ 8\ m)\ ps \wedge hd\ ps = (1,1) \wedge last\ ps = (7,2)$
  ⟨*proof*⟩

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

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

11 Knight's 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* ≡ *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*
  ⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

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* ≡ *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*
⟨*proof*⟩

abbreviation *b7x7* ≡ *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* ≡ *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*
⟨*proof*⟩

abbreviation *b7x9* ≡ *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* ≡ *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*
⟨*proof*⟩

abbreviation *b9x7* ≡ *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* ≡ *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*]])
lemma *kp-9x7-ul*: *knights-path b9x7 kp9x7ul*
⟨*proof*⟩

abbreviation *b9x9* ≡ *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* ≡ *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*
⟨*proof*⟩

The following lemma is a sub-proof used in Lemma 4 in [1]. I moved the sub-proof out to a separate lemma.

lemma *knights-circuit-exists-even-n-gr10*:
assumes *even n* $n \geq 10$ $m \geq 5$
$\exists ps.$ *knights-path* (*board* $(n-5)$ *m*) *ps* ∧ *hd ps* = (*int* $(n-5)$,*1*)
∧ *last ps* = (*int* $(n-5)-1$,*int* $m-1$)
shows $\exists ps.$ *knights-circuit* (*board m n*) *ps*
⟨*proof*⟩

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 [1]. 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 [1].

lemma *knights-path-odd-n-exists*:
  assumes *odd n min n m* $\geq$ *5*
  shows $\exists$*ps. knights-path* (*board n m*) *ps* $\wedge$ *hd ps* = (*int n*,*1*) $\wedge$ *last ps* = (*int n*−*1*,*int m*−*1*)
  ⟨*proof*⟩

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* $\geq$ *10* $\wedge$ *even n m* $\geq$ *5*
  shows $\exists$*ps. knights-circuit* (*board n m*) *ps*
  ⟨*proof*⟩

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 even* (*n*∗*m*)
  shows $\exists$*ps. knights-circuit* (*board n m*) *ps*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

THE END

end

References

[1] P. Cull and J. De Curtins. Knight's tour revisited. *Fibonacci Quarterly*, 16:276–285, 1978.