Latin Square

Alexander Bentkamp

March 17, 2025

Abstract

A theory about Latin Squares following [1]. A Latin Square is a $n \times n$ table filled with integers from 1 to n where each number appears exactly once in each row and each column. A Latin Rectangle is a partially filled $n \times n$ table with r filled rows and $n - r$ empty rows, such that each number appears at most once in each row and each column. The main result of this theory is that any Latin Rectangle can be completed to a Latin Square.

Contents

theory *Latin-Square*
imports *Marriage.Marriage*
begin

This theory is about Latin Squares. A Latin Square is a $n \times n$ table filled with integers from 1 to n where each number appears exactly once in each row and each column.

As described in "Das Buch der Beweise" a nice way to describe these squares by a $3 \times n$ matrix. Each column of this matrix contains the index of the row r, the index of the column c and the number in the cell (r,c). This $3 \times n$ matrix is called orthogonal array ("Zeilenmatrix").

I thought about different ways to formalize this orthogonal array, and came up with this: As the order of the columns in the array does not matter at all and no column can be a duplicate of another column, the orthogonal array is in fact a set of 3-tuples. Another advantage of formalizing it as a set is that it can easily model partially filled squares. For these 3-tuples I decided against 3-lists and against $nat \times nat \times nat$ (which is really $(nat \times nat) \times nat$) in favor of a function from a type with three elements to nat.

Additionally I use the numbers 0 to $n - 1$ instead of 1 to n for indexing the rows and columns as well as for filling the cells.

datatype *latin-type* = *Row* | *Col* | *Num* latin_type is of sort enum, needed for "value" command

```
instantiation latin-type :: enum
begin
 definition enum-latin-type == [Row, Col, Num]
 definition enum-all-latin-type (P:: latin-type ⇒ bool) = (P Row ∧ P Col ∧ P
Num)
 definition enum-ex-latin-type (P:: latin-type ⇒ bool) = (∃ x. P x)

instance
 apply standard
  apply (auto simp add: enum-latin-type-def enum-all-latin-type-def enum-ex-latin-type-def)
  apply (case-tac x,auto)
by (metis latin-type.exhaust)

end
```

Given a latin_type t, you might want to reference the other two. These are "next t" and "next (next t)":

```
definition [simp]:next t ≡ (case t of Row ⇒ Col | Col ⇒ Num | Num ⇒ Row)

lemma all-types-next-eqiv:(∀ t. P (next t)) ⟷ (∀ t. P t)
 apply (rule iffI)
  using next-def latin-type.case latin-type.exhaust apply metis
 apply metis
done
```

We call a column of the orthogonal array a latin_entry:

```
type-synonym latin-entry = latin-type ⇒ nat
```

This function removes one element of the 3-tupel and returns the other two as a pair:

```
definition without :: latin-type ⇒ latin-entry ⇒ nat × nat where
[simp]:without t ≡ λe. (e (next t), e (next (next t)))

value without Row (λt. case t of Row ⇒ 0 | Col ⇒ 1 | Num ⇒ 2) — returns
(1,2)

abbreviation row-col ≡ without Num
```

returns row and column of a latin_entry as a pair.

```
abbreviation col-num ≡ without Row
```

returns column and number of a latin_entry as a pair.

```
abbreviation num-row ≡ without Col
```

returns number and row of a latin_entry as a pair.

A partial latin square is a square that contains each number at most once in each row and each column, but not all cells have to be filled. Equivalently we can say that any two rows of the orthogonal array contain each pair of two numbers at most once. This can be expressed using the inj_on predicate:

definition *partial-latin-square* :: *latin-entry set* ⇒ *nat* ⇒ *bool* **where**
partial-latin-square s n ≡
(∀ *t*. *inj-on* (*without t*) *s*) ∧ — numbers are unique in each column (t=Row), numbers are unique in each row (t=Col), rows-column combinations are specified unambiguously (t=Num)
(∀ *e*∈*s*. ∀ *t*. *e t* < *n*) — all numbers, column indices and row indices are <n

value *partial-latin-square* {
(λ*t*. *case t of Row* ⇒ *0* | *Col* ⇒ *1* | *Num* ⇒ *0*),
(λ*t*. *case t of Row* ⇒ *1* | *Col* ⇒ *0* | *Num* ⇒ *1*)
} *2* — True

value *partial-latin-square* {
(λ*t*. *case t of Row* ⇒ *0* | *Col* ⇒ *0* | *Num* ⇒ *1*),
(λ*t*. *case t of Row* ⇒ *1* | *Col* ⇒ *0* | *Num* ⇒ *1*)
} *2* — False, because 1 appears twice in column 0

Looking at the orthogonal array a latin square is given iff any two rows of the orthogonal array contain each pair of two numbers at exactly once:

definition *latin-square* :: *latin-entry set* ⇒ *nat* ⇒ *bool* **where**
latin-square s n ≡
(∀ *t*. *bij-betw* (*without t*) *s* ({*0*..<*n*}×{*0*..<*n*}))

value *latin-square* {
(λ*t*. *case t of Row* ⇒ *0* | *Col* ⇒ *0* | *Num* ⇒ *1*), (λ*t*. *case t of Row* ⇒ *0* | *Col* ⇒ *1* | *Num* ⇒ *0*),
(λ*t*. *case t of Row* ⇒ *1* | *Col* ⇒ *0* | *Num* ⇒ *0*), (λ*t*. *case t of Row* ⇒ *1* | *Col* ⇒ *1* | *Num* ⇒ *1*)
} *2* — True

value *latin-square* {
(λ*t*. *case t of Row* ⇒ *0* | *Col* ⇒ *0* | *Num* ⇒ *1*), (λ*t*. *case t of Row* ⇒ *0* | *Col* ⇒ *1* | *Num* ⇒ *0*),
(λ*t*. *case t of Row* ⇒ *1* | *Col* ⇒ *0* | *Num* ⇒ *0*), (λ*t*. *case t of Row* ⇒ *1* | *Col* ⇒ *1* | *Num* ⇒ *0*)
} *2* — False, because 0 appears twice in Col 1 and twice in Row 1

A latin rectangle is a partial latin square in which the first m rows are filled and the following rows are empty:

definition *latin-rect* :: *latin-entry set* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
latin-rect s m n ≡
m ≤ *n* ∧
partial-latin-square s n ∧
bij-betw row-col s ({*0*..<*m*}×{*0*..<*n*}) ∧
bij-betw num-row s ({*0*..<*n*}×{*0*..<*m*})

value *latin-rect* {

($\lambda t.$ *case t of Row* $\Rightarrow$ *0* | *Col* $\Rightarrow$ *0* | *Num* $\Rightarrow$ *1*), ($\lambda t.$ *case t of Row* $\Rightarrow$ *0* | *Col* $\Rightarrow$ *1* | *Num* $\Rightarrow$ *0*)
} *1 2* — True

value *latin-rect* {
($\lambda t.$ *case t of Row* $\Rightarrow$ *0* | *Col* $\Rightarrow$ *0* | *Num* $\Rightarrow$ *1*), ($\lambda t.$ *case t of Row* $\Rightarrow$ *0* | *Col* $\Rightarrow$ *1* | *Num* $\Rightarrow$ *0*),
($\lambda t.$ *case t of Row* $\Rightarrow$ *1* | *Col* $\Rightarrow$ *0* | *Num* $\Rightarrow$ *0*), ($\lambda t.$ *case t of Row* $\Rightarrow$ *1* | *Col* $\Rightarrow$ *1* | *Num* $\Rightarrow$ *1*)
} *1 2* — False

There is another equivalent description of latin rectangles, which is easier to prove:

lemma *latin-rect-iff*:
$m \le n \wedge$ *partial-latin-square s n* $\wedge$ *card s* $= n * m \wedge (\forall e \in s.\ e\ Row < m) \longleftrightarrow$ *latin-rect s m n*
proof (*rule iffI*)
assume *prems*:$m \le n \wedge$ *partial-latin-square s n* $\wedge$ *card s* $= n * m \wedge (\forall e \in s.\ e\ Row < m)$

have *bij1*:*bij-betw row-col s* $(\{0..<m\} \times \{0..<n\})$ **using** *prems*
proof
have *inj-on row-col s* **using** *prems partial-latin-square-def* **by** *blast*
moreover have $\{0..<m\} \times \{0..<n\}$ = *row-col ' s*
proof−
have *row-col ' s* $\subseteq \{0..<m\} \times \{0..<n\}$ **using** *prems partial-latin-square-def* **by** *auto*
moreover have *card* (*row-col ' s*) = *card* $(\{0..<m\} \times \{0..<n\})$ **using** *prems card-image*[*OF* ‹*inj-on row-col s*›] **by** *auto*
ultimately show $\{0..<m\} \times \{0..<n\}$ = *row-col ' s* **using** *card-subset-eq*[*of* $\{0..<m\} \times \{0..<n\}$ *row-col ' s*] **by** *auto*
qed
ultimately show *?thesis* **unfolding** *bij-betw-def* **by** *auto*
qed

have *bij2*:*bij-betw num-row s* $(\{0..<n\} \times \{0..<m\})$ **using** *prems*
proof
have *inj-on num-row s* **using** *prems partial-latin-square-def* **by** *blast*
moreover have $\{0..<n\} \times \{0..<m\}$ = *num-row ' s*
proof−
have *num-row ' s* $\subseteq \{0..<n\} \times \{0..<m\}$ **using** *prems partial-latin-square-def* **by** *auto*
moreover have *card* (*num-row ' s*) = *card* $(\{0..<n\} \times \{0..<m\})$ **using** *prems card-image*[*OF* ‹*inj-on num-row s*›] **by** *auto*
ultimately show $\{0..<n\} \times \{0..<m\}$ = *num-row ' s* **using** *card-subset-eq*[*of* $\{0..<n\} \times \{0..<m\}$ *num-row ' s*] **by** *auto*
qed
ultimately show *?thesis* **unfolding** *bij-betw-def* **by** *auto*
qed

from *prems bij1 bij2* **show** *latin-rect s m n* **unfolding** *latin-rect-def* **by** *auto*
next
assume *prems*:*latin-rect s m n*
have *m*≤*n partial-latin-square s n* **using** *latin-rect-def prems* **by** *auto*
moreover have *card s* = *m* ∗ *n*
proof −
have *bij-betw row-col s* ({*0*..<*m*} × {*0*..<*n*}) **using** *latin-rect-def prems* **by** *auto*
then show *?thesis* **using** *bij-betw-same-card*[*of row-col s* {*0*..<*m*} × {*0*..<*n*}] **by** *auto*
qed
moreover have ∀ *e*∈*s*. *e Row* < *m* **using** *latin-rect-def prems* **using** *atLeast0LessThan bij-betwE* **by** *fastforce*
ultimately show *m*≤*n* ∧ *partial-latin-square s n* ∧ *card s* = *n* ∗ *m* ∧ (∀ *e*∈*s*. *e Row* < *m*) **by** *auto*
qed

A square is a latin square iff it is a partial latin square with all n^2 cells filled:

lemma *partial-latin-square-full*:
partial-latin-square s n ∧ *card s* = *n*∗*n* ⟷ *latin-square s n*
proof (*rule iffI*)
assume *prem*: *partial-latin-square s n* ∧ *card s* = *n* ∗ *n*
have ∀ *t*. (*without t*) ‘ *s* ⊆ {*0*..<*n*} × {*0*..<*n*}
proof
fix *t* **show** (*without t*) ‘ *s* ⊆ {*0*..<*n*} × {*0*..<*n*} **using** *partial-latin-square-def next-def atLeast0LessThan prem* **by** (*cases t*) *auto*
qed
then show *partial-latin-square s n* ∧ *card s* = *n* ∗ *n* ⟹ *latin-square s n*
unfolding *latin-square-def* **using** *partial-latin-square-def*
by (*metis bij-betw-def card-atLeastLessThan card-cartesian-product card-image card-subset-eq diff-zero finite-SigmaI finite-atLeastLessThan*)
next
assume *prem*:*latin-square s n*
then have *bij-betw row-col s* ({*0*..<*n*} × {*0*..<*n*}) **using** *latin-square-def* **by** *blast*
moreover have *partial-latin-square s n*
proof −
have ∀ *t*. ∀ *e*∈*s*. (*without t*) *e* ∈ ({*0*..<*n*}×{*0*..<*n*}) **using** *prem latin-square-def bij-betwE* **by** *metis*
then have *1*:∀ *e*∈*s*.∀ *t*. *e t* < *n* **using** *latin-square-def all-types-next-eqiv*[*of* λ*t*. ∀ *e*∈*s*. *e t* < *n*] *bij-betwE* **by** *auto*
have *2*:(∀ *t*. *inj-on* (*without t*) *s*) **using** *prem bij-betw-def latin-square-def* **by** *auto*
from *1 2* **show** *?thesis* **using** *partial-latin-square-def* **by** *auto*
qed
ultimately show *partial-latin-square s n* ∧ *card s* = *n*∗*n* **by** (*auto simp add*: *bij-betw-same-card*)

qed

Now we prove Lemma 1 from chapter 27 in "Das Buch der Beweise". But first some lemmas, that prove very intuitive facts:

lemma *bij-restrict*:
assumes *bij-betw f A B* ∀ *a*∈*A*. *P a* ⟷ *Q* (*f a*)
shows *bij-betw f* {*a*∈*A*. *P a*} {*b*∈*B*. *Q b*}
proof −
  have *inj*: *inj-on f* {*a*∈*A*. *P a*} **using** *assms bij-betw-def* **by** (*metis* (*mono-tags*, *lifting*) *inj-onD inj-onI mem-Collect-eq*)
  have *surj1*: *f* ‘ {*a*∈*A*. *P a*} ⊆ {*b*∈*B*. *Q b*} **using** *assms*(*1*) *assms*(*2*) *bij-betwE* **by** *blast*
  have *surj2*: {*b*∈*B*. *Q b*} ⊆ *f* ‘ {*a*∈*A*. *P a*}
  proof
    fix *b*
    assume *b* ∈ {*b* ∈ *B*. *Q b*}
    then obtain *a* **where** *f a* = *b a*∈*A* **using** *assms*(*1*) *bij-betw-inv-into-right bij-betwE bij-betw-inv-into mem-Collect-eq* **by** (*metis* (*no-types*, *lifting*))
    then show *b* ∈ *f* ‘ {*a*∈*A*. *P a*} **using** ‹*b* ∈ {*b* ∈ *B*. *Q b*}› *assms*(*2*) **by** *blast*
  qed
  with *inj surj1 surj2* **show** *?thesis* **using** *bij-betw-imageI* **by** *fastforce*
qed

lemma *cartesian-product-margin1*:
assumes *a*∈*A*
shows {*p*∈*A*×*B*. *fst p* = *a*} = {*a*}×*B*
using *SigmaI assms* **by** *auto*

lemma *cartesian-product-margin2*:
assumes *b*∈*B*
shows {*p*∈*A*×*B*. *snd p* = *b*} = *A*×{*b*}
using *SigmaI assms* **by** *auto*

The union of sets containing at most k elements each cannot contain more elements than the number of sets times k:

lemma *limited-family-union*: *finite B* ⟹ ∀ *P*∈*B*. *card P* ≤ *k* ⟹ *card* (⋃*B*) ≤ *card B* ∗ *k*
proof (*induction B rule*:*finite-induct*)
  case *empty*
  then show *?case* **by** *auto*
next
  case (*insert P B*)
  have *card* (⋃(*insert P B*)) ≤ *card P* + *card* (⋃*B*) **by** (*simp add*: *card-Un-le*)
  then have *card* (⋃(*insert P B*)) ≤ *card P* + *card B* ∗ *k* **using** *insert* **by** *auto*
  then show *?case* **using** *insert* **by** *simp*
qed

If f hits each element at most k times, the domain of f can only be k times bigger than the image of f:

lemma *limited-preimages*:
assumes $\forall\, x \in f$ ‘ D. *card* $((f -‘ \{x\}) \cap D) \leq k$ *finite D*
shows *card* $D \leq$ *card* $(f$ ‘ $D) * k$
proof −
  let *?preimages* = $(\lambda x.\ (f -‘ \{x\}) \cap D)$ ‘ $(f$ ‘ $D)$
  have $D = \bigcup$ *?preimages* **by** *auto*
  have *card* $(\bigcup$ *?preimages*$) \leq$ *card ?preimages* $* k$ **using** *limited-family-union*[*of ?preimages k*] *assms* **by** *auto*
  moreover have *card* (*?preimages*) $* k \leq$ *card* $(f$ ‘ $D) * k$ **using** *card-image-le*[*of f* ‘ D $\lambda x.\ (f -‘ \{x\}) \cap D$] *assms* **by** *auto*
  ultimately have *card* $(\bigcup$ *?preimages*$) \leq$ *card* $(f$ ‘ $D) * k$ **using** *le-trans* **by** *blast*
  then show *?thesis* **using** ‹$D = \bigcup$ *?preimages*› **by** *metis*
qed

Let $A_1, \ldots, A_n$ be sets with $k > 0$ elements each. Any element is only contained in at most k of these sets. Then there are more different elements in total than sets A_i:

lemma *union-limited-replicates*:
assumes *finite I* $\forall\, i{\in}I$. *finite* $(A\ i)$ $k{>}0$ $\forall\, i{\in}I$. *card* $(A\ i) = k$ $\forall\, i{\in}I.\ \forall\, x{\in}(A\ i)$. *card* $\{i{\in}I.\ x{\in}A\ i\} \leq k$
shows *card* $(\bigcup i{\in}I.\ (A\ i)) \geq$ *card I* **using** *assms*
proof −
  let *?pairs* = $\{(i,x).\ i{\in}I \wedge x{\in}A\ i\}$

  have *card-pairs*: *card ?pairs* = *card I* $* k$ **using** *assms*
  proof (*induction I rule*:*finite-induct*)
    case *empty*
    then show *?case* **using** *card-eq-0-iff* **by** *auto*
  next
    case (*insert i0 I*)
    have $\forall\, i{\in}I.\ \forall\, x{\in}(A\ i)$. *card* $\{i{\in}I.\ x{\in}A\ i\} \leq k$
    proof (*rule ballI*)+
      fix *i x* **assume** $i \in I$ $x{\in}A\ i$
      then have *card* $\{i \in$ *insert i0 I*. $x \in A\ i\} \leq k$ **using** *insert* **by** *auto*
      moreover have *finite* $\{i \in$ *insert i0 I*. $x \in A\ i\}$ **using** *insert* **by** *auto*
      ultimately show *card* $\{i{\in}I.\ x{\in}A\ i\} \leq k$ **using** *card-mono*[*of* $\{i \in$ *insert i0 I*. $x \in A\ i\}$ $\{i \in I.\ x \in A\ i\}$] *le-trans* **by** *blast*
    qed
    then have *card-S*: *card* $\{(i, x).\ i \in I \wedge x \in A\ i\}$ = *card I* $* k$ **using** *insert* **by** *auto*

    have *card-B*: *card* $\{(i, x).\ i{=}i0 \wedge x{\in}A\ i0\} = k$ **using** *insert* **by** *auto*

    have $\{(i, x).\ i \in$ *insert i0 I* $\wedge x \in A\ i\} = \{(i, x).\ i \in I \wedge x \in A\ i\} \cup \{(i, x).\ i{=}i0 \wedge x{\in}A\ i0\}$ **by** *auto*
    moreover have $\{(i, x).\ i \in I \wedge x \in A\ i\} \cap \{(i, x).\ i{=}i0 \wedge x{\in}A\ i0\} = \{\}$ **using** *insert* **by** *auto*
    moreover have *finite* $\{(i, x).\ i \in I \wedge x \in A\ i\}$ **using** *insert rev-finite-subset*[*of* $I \times \bigcup(A$ ‘ $I)$ $\{(i, x).\ i \in I \wedge x \in A\ i\}$] **by** *auto*

moreover have *finite* $\{(i, x).\ i{=}i0 \wedge x{\in}A\ i0\}$ **using** *insert card-B card.infinite neq0-conv* **by** *blast*

ultimately have *card* $\{(i, x).\ i \in insert\ i0\ I \wedge x \in A\ i\} = card\ \{(i, x).\ i \in I \wedge x \in A\ i\} + card\ \{(i, x).\ i{=}i0 \wedge x{\in}A\ i0\}$ **by** (*simp add*: *card-Un-disjoint*)

with *card-S card-B* **have** *card* $\{(i, x).\ i \in insert\ i0\ I \wedge x \in A\ i\} = (card\ I + 1) * k$ **by** *auto*

then show *?case* **using** *insert* **by** *auto*

qed

define *f* **where** $f\ ix = (case\ ix\ of\ (i,x) \Rightarrow x)$ **for** $ix :: 'a \times 'b$

have *preimages-le-k*: $\forall x \in f\ `\ ?pairs.\ card\ ((f\ -`\ \{x\}) \cap ?pairs) \leq k$

proof

fix *x0* **assume** *x0-def*: $x0 \in f\ `\ ?pairs$

have $(f\ -`\ \{x0\}) \cap ?pairs = \{(i,x).\ i{\in}I \wedge x{\in}A\ i \wedge x{=}x0\}$ **using** *f-def* **by** *auto*

moreover have *card* $\{(i,x).\ i{\in}I \wedge x{\in}A\ i \wedge x{=}x0\} = card\ \{i{\in}I.\ x0{\in}A\ i\}$ **using** ‹*finite I*›

proof −

have *inj-on* $(\lambda i.\ (i,x0))\ \{i{\in}I.\ x0{\in}A\ i\}$ **by** (*meson Pair-inject inj-onI*)

moreover have $(\lambda i.\ (i,x0))\ `\ \{i{\in}I.\ x0{\in}A\ i\} = \{(i,x).\ i{\in}I \wedge x{\in}A\ i \wedge x{=}x0\}$ **by** (*rule subset-antisym*) *blast+*

ultimately show *?thesis* **using** *card-image* **by** *fastforce*

qed

ultimately have *1*:*card* $((f\ -`\ \{x0\}) \cap ?pairs) = card\ \ \{i{\in}I.\ x0{\in}A\ i\}$ **by** *auto*

have $\exists\ i0.\ x0{\in}A\ i0 \wedge i0{\in}I$ **using** *x0-def f-def* **by** *auto*

then have *card* $\{i{\in}I.\ x0{\in}A\ i\} \leq k$ **using** *assms* **by** *auto*

with *1* **show** *card* $((f\ -`\ \{x0\}) \cap ?pairs) \leq k$ **by** *auto*

qed

have *card* $?pairs \leq card\ (f\ `\ ?pairs) * k$

proof −

have *finite* $\{(i, x).\ i \in I \wedge x \in A\ i\}$ **using** *assms card-pairs not-finite-existsD* **by** *fastforce*

then show *?thesis* **using** *limited-preimages*[*of f ?pairs k*, *OF preimages-le-k*] **by** *auto*

qed

then have *card* $I \leq card\ (f\ `\ ?pairs)$ **using** *card-pairs assms* **by** *auto*

moreover have $f\ `\ ?pairs = (\bigcup i{\in}I.\ (A\ i))$ **using** *f-def* [*abs-def*] **by** *auto*

ultimately show *?thesis* **using** *f-def* **by** *auto*

qed

In a $m \times n$ latin rectangle each number appears in m columns:

lemma *latin-rect-card-col*:

assumes *latin-rect s m n* $x{<}n$

shows *card* $\{e\ Col|e.\ e{\in}s \wedge e\ Num = x\} = m$

```
proof -
  have card {e ∈ s. e Num = x} = m
  proof -
    have 1:bij-betw num-row s ({0..<n}×{0..<m}) using assms latin-rect-def by auto
    have 2:∀ e∈s. e Num = x ⟷ fst (num-row e) = x by simp
    have bij-betw num-row {e∈s. e Num = x} ({x}×{0..<m})
      using bij-restrict[OF 1 2] cartesian-product-margin1[of x {0..<n} {0..<m}] assms by auto
    then show ?thesis using card-cartesian-product by (simp add: bij-betw-same-card)
  qed
  moreover have card {e∈s. e Num = x} = card {e Col |e. e ∈ s ∧ e Num = x}
  proof -
    have inj-on col-num s using assms latin-rect-def[of s m n] partial-latin-square-def[of s n] by blast
    then have inj-on col-num {e∈s. e Num = x} by (metis (mono-tags, lifting) inj-onD inj-onI mem-Collect-eq)
    then have inj-on (λe. e Col) {e∈s. e Num = x} unfolding inj-on-def using without-def by auto
    moreover have (λe. e Col) ` {e∈s. e Num = x} = {e Col |e. e ∈ s ∧ e Num = x} by (rule subset-antisym) blast+
    ultimately show ?thesis using card-image by fastforce
  qed
  ultimately show ?thesis by auto
qed
```

In a $m \times n$ latin rectangle each column contains m numbers:

```
lemma latin-rect-card-num:
assumes latin-rect s m n x<n
shows card {e Num|e. e∈s ∧ e Col = x} = m
proof -
  have card {e ∈ s. e Col = x} = m
  proof -
    have 1:bij-betw row-col s ({0..<m}×{0..<n}) using assms latin-rect-def by auto
    have 2:∀ e∈s. e Col = x ⟷ snd (row-col e) = x by simp
    have bij-betw row-col {e∈s. e Col = x} ({0..<m}×{x})
      using bij-restrict[OF 1 2] cartesian-product-margin2[of x {0..<n} {0..<m}] assms by auto
    then show ?thesis using card-cartesian-product by (simp add: bij-betw-same-card)
  qed
  moreover have card {e∈s. e Col = x} = card {e Num |e. e ∈ s ∧ e Col = x}
  proof -
    have inj-on col-num s using assms latin-rect-def[of s m n] partial-latin-square-def[of s n] by blast
    then have inj-on col-num {e∈s. e Col = x} by (metis (mono-tags, lifting) inj-onD inj-onI mem-Collect-eq)
    then have inj-on (λe. e Num) {e∈s. e Col = x} unfolding inj-on-def using without-def by auto
```

moreover have $(\lambda e.\ e\ Num)$ ‘ $\{e{\in}s.\ e\ Col = x\} = \{e\ Num\ |e.\ e \in s \wedge e\ Col = x\}$ **by** (*rule subset-antisym*) *blast+*
ultimately show *?thesis* **using** *card-image* **by** *fastforce*
qed
ultimately show *?thesis* **by** *auto*
qed

Finally we prove lemma 1 chapter 27 of "Das Buch der Beweise":

theorem
assumes *latin-rect s* $(n{-}m)$ *n* $m{\le}n$
shows $\exists\, s'.\ s{\subseteq}s' \wedge$ *latin-square* s' *n*
using *assms*
proof (*induction m arbitrary*:*s*) — induction over the number of empty rows
case *0*
then have *bij-betw row-col s* $(\{0..{<}n\} \times \{0..{<}n\})$ **using** *latin-rect-def* **by** *auto*
then have *card s* $= n{*}n$ **by** (*simp add*:*bij-betw-same-card*)
then show *?case* **using** *partial-latin-square-full 0 latin-rect-def* **by** *auto*
next
case (*Suc m*)

— We use the Hall theorem on the sets A_j of numbers that do not occur in column j:
let *?not-in-column* $= \lambda j.\ \{0..{<}n\} - \{e\ Num\ |e.\ e{\in}s \wedge e\ Col = j\}$

— Proof of the hall condition:
have $\forall\, J{\subseteq}\{0..{<}n\}.$ *card J* $\le$ *card* $(\bigcup j{\in}J.$ *?not-in-column j*$)$
proof (*rule allI*; *rule impI*)
fix *J* **assume** *J-def*:$J{\subseteq}\{0..{<}n\}$
have $\forall\, j{\in}J.$ *card* (*?not-in-column j*) = *Suc m*
proof
fix *j* **assume** *j-def*:$j{\in}J$
have $\{e\ Num\ |e.\ e{\in}s \wedge e\ Col = j\} \subseteq \{0..{<}n\}$ **using** *atLeastLessThan-iff Suc latin-rect-def partial-latin-square-def* **by** *auto*
moreover then have *finite* $\{e\ Num\ |e.\ e{\in}s \wedge e\ Col = j\}$ **using** *finite-subset* **by** *auto*
ultimately have *card* (*?not-in-column j*) = *card* $\{0..{<}n\}$ − *card* $\{e\ Num\ |e.\ e \in s \wedge e\ Col = j\}$ **using** *card-Diff-subset*[*of* $\{e\ Num\ |e.\ e{\in}s \wedge e\ Col = j\}$ $\{0..{<}n\}$] **by** *auto*
then show *card*(*?not-in-column j*) = *Suc m* **using** *latin-rect-card-num J-def j-def Suc* **by** *auto*
qed
moreover have $\forall\, j0{\in}J.\ \forall\, x{\in}$*?not-in-column j0*. *card* $\{j \in J.\ x \in$ *?not-in-column j*$\} \le$ *Suc m*
proof (*rule ballI*; *rule ballI*)
fix *j0 x* **assume** $j0 \in J$ $x \in$ *?not-in-column j0*
then have *card* $(\{0..{<}n\} - \{e\ Col|e.\ e{\in}s \wedge e\ Num = x\})$ = *Suc m*
proof −
have *card* $\{e\ Col|e.\ e{\in}s \wedge e\ Num = x\} = n -$ *Suc m* **using** *latin-rect-card-col* ‹$x \in$ *?not-in-column j0*› *Suc* **by** *auto*

moreover have $\{e\ Col|e.\ e{\in}s \wedge e\ Num = x\}{\subseteq}\{0..{<}n\}$ **using** *Suc latin-rect-def partial-latin-square-def* **by** *auto*
moreover then have *finite* $\{e\ Col|e.\ e{\in}s \wedge e\ Num = x\}$ **using** *finite-subset* **by** *auto*
ultimately show *?thesis* **using** *card-Diff-subset*[*of* $\{e\ Col|e.\ e{\in}s \wedge e\ Num = x\}$ $\{0..{<}n\}$] **using** *Suc.prems* **by** *auto*
qed
moreover have $\{j \in J.\ x \in ?not\text{-}in\text{-}column\ j\} \subseteq \{0..{<}n\} - \{e\ Col|e.\ e{\in}s \wedge e\ Num = x\}$ **using** *Diff-mono J-def* **using** ‹$x \in ?not\text{-}in\text{-}column\ j0$› **by** *blast*
ultimately show *card* $\{j \in J.\ x \in ?not\text{-}in\text{-}column\ j\} \leq Suc\ m$ **by** (*metis* (*no-types*, *lifting*) *card-mono finite-Diff finite-atLeastLessThan*)
qed
moreover have *finite J* **using** *J-def finite-subset* **by** *auto*
ultimately show *card J* $\leq$ *card* ($\bigcup j{\in}J.\ ?not\text{-}in\text{-}column\ j$) **using** *union-limited-replicates*[*of J ?not-in-column Suc m*] **by** *auto*
qed

— The Hall theorem gives us a system of distinct representatives, which we can use to fill the next row:
then obtain *R* **where** *R-def*:$\forall j{\in}\{0..{<}n\}.\ R\ j \in ?not\text{-}in\text{-}column\ j \wedge inj\text{-}on\ R\ \{0..{<}n\}$ **using** *marriage-HV*[*of* $\{0..{<}n\}$ *?not-in-column*] **by** *blast*

define *new-row* **where** *new-row* = ($\lambda j.$ *rec-latin-type* ($n - Suc\ m$) *j* (*R j*)) ‘ $\{0..{<}n\}$
define s' **where** $s' = s \cup new\text{-}row$

— s’ is now a latin rect with one more row:
have *latin-rect* s' $(n{-}m)$ *n*
proof −
— We prove all four criteria specified in the lemma latinrectiff:
have $n{-}m \leq n$ **by** *auto*
moreover have *partial-latin-square* s' *n*
proof −
have *inj-on* (*without Col*) s' **unfolding** *inj-on-def*
proof (*rule ballI*; *rule ballI*; *rule impI*)
fix *e1 e2* **assume** $e1 \in s'$ $e2 \in s'$ *num-row e1* = *num-row e2*
then have *e1 Num* = *e2 Num e1 Row* = *e2 Row* **using** *without-def* **by** *auto*
moreover have *e1 Col* = *e2 Col*
proof (*cases*)
assume *e1 Row* = $n - Suc\ m$
then have *e2 Row* = $n - Suc\ m$ **using** *without-def* ‹*num-row e1* = *num-row e2*› **by** *auto*
have $\forall e{\in}s.\ e\ Row < n - Suc\ m$ **using** *Suc latin-rect-iff* **by** *blast*
then have $e1 \in new\text{-}row$ $e2 \in new\text{-}row$ **using** s'*-def* ‹$e1 \in s'$› ‹$e2 \in s'$› ‹$e1\ Row = n - Suc\ m$› ‹$e2\ Row = n - Suc\ m$› **by** *auto*
then have *e1 Num* = *R* (*e1 Col*) *e2 Num* = *R* (*e2 Col*) **using** *new-row-def* **by** *auto*
then have *R* (*e1 Col*) = *R* (*e2 Col*) **using** ‹*e1 Num* = *e2 Num*› **by** *auto*
moreover have $e1\ Col < n$ $e2\ Col < n$ **using** ‹$e1 \in new\text{-}row$› ‹$e2 \in$

new-row› *new-row-def* **by** *auto*
ultimately show *e1 Col* = *e2 Col* **using** *R-def inj-on-def* **by** (*metis* (*mono-tags*, *lifting*) *atLeast0LessThan lessThan-iff*)
next
assume *e1 Row* ≠ *n* − *Suc m*
then have *e1*∈*s e2*∈*s* **using** *new-row-def s′-def* ‹*e1*∈*s′*› ‹*e2*∈*s′*› ‹*e1 Row* = *e2 Row*› **by** *auto*
then show *e1 Col* = *e2 Col* **using** *Suc latin-rect-def bij-betw-def* **by** (*metis* ‹*num-row e1* = *num-row e2*› *inj-onD*)
qed
ultimately show *e1*=*e2* **using** *latin-type.induct*[*of* λ*t*. *e1 t* = *e2 t*] **by** *auto*
qed
moreover have *inj-on* (*without Row*) *s′* **unfolding** *inj-on-def*
proof (*rule ballI*; *rule ballI*; *rule impI*)
fix *e1 e2* **assume** *e1* ∈ *s′ e2* ∈ *s′ col-num e1* = *col-num e2*
then have *e1 Col* = *e2 Col e1 Num* = *e2 Num* **using** *without-def* **by** *auto*
moreover have *e1 Row* = *e2 Row*
proof (*cases*)
assume *e1 Row* = *n* − *Suc m*
have ∀ *e*∈*s*. *e Row* < *n* − *Suc m* **using** *Suc latin-rect-iff* **by** *blast*
then have *e2 Num* ∈ *?not-in-column* (*e2 Col*) **using** *R-def new-row-def* ‹*e1 Col* = *e2 Col*› ‹*e1 Num* = *e2 Num*› **using** *s′-def* ‹*e1* ∈ *s′*› ‹*e1 Row* = *n* − *Suc m*› **by** *auto*
then show *e1 Row* = *e2 Row* **using** *new-row-def* ‹*e1 Row* = *n* − *Suc m*› *s′-def* ‹*e2* ∈ *s′*› **by** *auto*
next
assume *e1 Row* ≠ *n* − *Suc m*
then have *e1*∈*s* **using** *new-row-def s′-def* ‹*e1*∈*s′*› **by** *auto*
then have *e2 Num* ∉ *?not-in-column* (*e2 Col*) **using** ‹*e1 Col* = *e2 Col*› ‹*e1 Num* = *e2 Num*› **by** *auto*
then have *e2*∈*s* **using** *new-row-def s′-def* ‹*e2*∈*s′*› *R-def* **by** *auto*
moreover have *inj-on col-num s* **using** *Suc.prems latin-rect-def*[*of s* (*n* − *Suc m*) *n*] *partial-latin-square-def*[*of s n*] **by** *blast*
ultimately show *e1 Row* = *e2 Row* **using** *Suc latin-rect-def* **by** (*metis* ‹*col-num e1* = *col-num e2*› ‹*e1* ∈ *s*› *inj-onD*)
qed
ultimately show *e1*=*e2* **using** *latin-type.induct*[*of* λ*t*. *e1 t* = *e2 t*] **by** *auto*
qed
moreover have *inj-on* (*without Num*) *s′* **unfolding** *inj-on-def*
proof (*rule ballI*; *rule ballI*; *rule impI*)
fix *e1 e2* **assume** *e1* ∈ *s′ e2* ∈ *s′ row-col e1* = *row-col e2*
then have *e1 Row* = *e2 Row e1 Col* = *e2 Col* **using** *without-def* **by** *auto*
moreover have *e1 Num* = *e2 Num*
proof (*cases*)
assume *e1 Row* = *n* − *Suc m*
then have *e2 Row* = *n* − *Suc m* **using** *without-def* ‹*row-col e1* = *row-col e2*› **by** *auto*

have ∀ $e \in s$. *e Row* < *n* − *Suc m* **using** *Suc latin-rect-iff* **by** *blast*
then show *e1 Num* = *e2 Num* **using** ‹*e1 Col* = *e2 Col*› **using** *new-row-def s'-def* ‹*e1* ∈ *s'*› ‹*e2* ∈ *s'*› ‹*e1 Row* = *n* − *Suc m*› ‹*e2 Row* = *n* − *Suc m*› **by** *auto*
next
assume *e1 Row* ≠ *n* − *Suc m*
then have *e1*∈*s* *e2*∈*s* **using** *new-row-def s'-def* ‹*e1*∈*s'*› ‹*e2*∈*s'*› ‹*e1 Row* = *e2 Row*› **by** *auto*
then show *e1 Num* = *e2 Num* **using** *Suc latin-rect-def bij-betw-def* **by** (*metis* ‹*row-col e1* = *row-col e2*› *inj-onD*)
qed
ultimately show *e1*=*e2* **using** *latin-type.induct*[*of* λ*t*. *e1 t* = *e2 t*] **by** *auto*
qed
moreover have ∀ *e*∈*s'*. ∀ *t*. *e t* < *n*
proof (*rule ballI*; *rule allI*)
fix *e t* **assume** *e*∈*s'*
then show *e t* < *n*
proof (*cases*)
assume *e*∈*new-row*
then show *?thesis* **using** *new-row-def R-def* **by** (*induction t*) *auto*
next
assume *e*∉*new-row*
then show *?thesis* **using** *s'-def* ‹*e*∈*s'*› *latin-rect-def partial-latin-square-def Suc* **by** *auto*
qed
qed
ultimately show *partial-latin-square s' n* **unfolding** *partial-latin-square-def* **using** *latin-type.induct*[*of* λ*t*. *inj-on* (*without t*) *s'*] **by** *auto*
qed
moreover have *card s'* = *n* ∗ (*n* − *m*)
proof −
have *card-s*:*card s* = *n* ∗ (*n* − *Suc m*) **using** *latin-rect-iff Suc* **by** *auto*
have *card-new-row*:*card new-row* = *n* **unfolding** *new-row-def*
proof −
have *inj-on* (λ*j*. *rec-latin-type* (*n* − *Suc m*) *j* (*R j*)) {*0*..<*n*} **unfolding** *inj-on-def*
proof (*rule ballI*; *rule ballI*; *rule impI*)
fix *j1 j2* **assume** *j1* ∈ {*0*..<*n*} *j2* ∈ {*0*..<*n*} *rec-latin-type* (*n* − *Suc m*) *j1* (*R j1*) = *rec-latin-type* (*n* − *Suc m*) *j2* (*R j2*)
then show *j1* = *j2* **using** *latin-type.rec*(*2*)[*of* (*n* − *Suc m*) *j1 R j1*] *latin-type.rec*(*2*)[*of - j2 -*] **by** *auto*
qed
then show *card* ((λ*j*. *rec-latin-type* (*n* − *Suc m*) *j* (*R j*)) ‘ {*0*..<*n*}) = *n* **by** (*simp add*: *card-image*)
qed
have *s* ∩ *new-row* = {}
proof −
have ∀ *e*∈*s*. *e Row* < *n* − *Suc m* **using** *Suc latin-rect-iff* **by** *blast*
then have ∀ *e* ∈ *new-row*. *e* ∉ *s* **using** *new-row-def* **by** *auto*

then show *?thesis* **by** *blast*
qed
moreover have *finite s* **using** *Suc latin-rect-def* **by** (*metis bij-betw-finite finite-SigmaI finite-atLeastLessThan*)
moreover have *finite new-row* **using** *new-row-def* **by** *simp*
ultimately have *card s′ = card s + card new-row* **using** *s′-def card-Un-disjoint* **by** *auto*
with *card-s card-new-row* **show** *?thesis* **using** *Suc* **by** (*metis Suc-diff-Suc Suc-le-lessD add.commute mult-Suc-right*)
qed
moreover have ∀ *e*∈*s′*. *e Row* < (*n* − *m*)
proof (*rule ballI*; *cases*)
fix *e*
assume *e*∈*new-row*
then show *e Row* < *n* − *m* **using** *Suc new-row-def R-def* **by** *auto*
next
fix *e*
assume *e* ∈ *s′* *e*∉*new-row*
then have *e Row* < *n* − *Suc m* **using** *latin-rect-iff Suc s′-def* ‹*e*∈*s′*› **by** *auto*
then show *e Row* < *n* − *m* **by** *auto*
qed
ultimately show *?thesis* **using** *latin-rect-iff*[*of n*−*m n*] **by** *auto*
qed

— Finally we use the induction hypothesis:
then obtain *s″* **where** *s′* ⊆ *s″* *latin-square s″ n* **using** *Suc* **by** *auto*
then have *s* ⊆ *s″* **using** *s′-def* **by** *auto*
then show ∃ *s′*. *s* ⊆ *s′* ∧ *latin-square s′ n* **using** ‹*latin-square s″ n*› **by** *auto*
qed

end

References

[1] M. Aigner and G. Ziegler. *Das Buch der Beweise*. Springer, 2004.