

Countable Infinite Version of Dilworth’s Theorem

Fabián Fernando Serrano Suárez[†], Thaynara Arielly de Lima^{*},
Mauricio Ayala-Rincón^{*,‡}

[†] Universidad Nacional de Colombia - Sede Manizales, Colombia

[‡]Universidade de Brasília, Brasília D.F., Brazil

^{*}Universidade Federal de Goiás, Goiânia, Brazil

October 26, 2026

Abstract

Dilworth’s Theorem states that in any finite partially ordered set, the size of a largest antichain equals the minimum number of chains needed to cover the set. We report a complete, machine-checked formalization of Dilworth’s Theorem and of its extension to countable partial orders of finite width in the Isabelle/HOL proof assistant. The finite case is obtained from a formalization of the König–Egerváry Theorem on the construction of a bipartite digraph associated with the partial order. For finite width, the countable case reduces to the finite one through a compactness argument: the incomparability graph of the partial order is shown to be k -colorable whenever every finite induced subgraph is, by appealing to a formalization of the De Bruijn–Erdős coloring theorem, itself a consequence of the compactness of propositional logic. We describe the formal statements, the central definitions, and the two top-level theorems together with their Isar proofs. This development continues the previous work on the mechanization of combinatorial theorems over countable infinite structures derived from propositional compactness.

Keywords: Dilworth’s Theorem · Isabelle/HOL · Partial Order Theory · Chains and Antichains · De Bruijn–Erdős Theorem · König–Egerváry Theorem · Propositional Compactness.

1 Introduction

In the PhD thesis, an Isabelle/HOL formalization of the theory of propositional logic that includes landmark theorems such as the compactness theorem was developed [11], further adapted and included in the Isabelle theory on compactness and combinatorial applications [15]. This theory includes formalizations of combinatorial theorems extended to the countable (infinite) case though the constructive application of the compactness theorem,

which include the first formalization of the countable version of Hall’s Theorem (cf. [4]) for countable families of sets, the countable graph-version of Hall’s Theorem, as well as the de Bruijn–Erdős k -coloring theorem (cf. [2]) for infinite graphs [12, 13]. The formalizations use standard compactness arguments, building models for finite substructures from which the combinatorial theorem for infinite structures follows. For instance, the formalization of the infinite version of Hall’s Theorem uses the completeness theorem and the finite formalization of Hall’s Theorem developed by Jiang and Nipkow [5, 6].

Following this line of research, this paper addresses the countable version of Dilworth’s Theorem for partial orders of *finite width*, a case that — in contrast to the situation for infinite width discussed below — does follow from the finite theorem by a standard compactness argument; our contribution is a complete machine-checked formalization of it. A *chain* in a partially ordered set $P = (A, \leq)$ is a subset that is totally ordered by $\leq$, and an *antichain* is a subset of pairwise incomparable elements. Dilworth’s Theorem [3] asserts that, for a finite P , the maximum cardinality of an antichain (the *width* of P) equals the least number of chains into which A can be partitioned. The result is a cornerstone of order theory and combinatorics and admits several classical proofs, including one through the König-Egerváry Theorem on bipartite graphs. Indeed, also in his seminal paper [3], Dilworth proved the general theorem that if an infinite poset has a finite maximum antichain size w , it can be partitioned into exactly w chains. Dilworth did not use a standard compactness approach. Instead, he found an ad hoc proof that constructed the infinite decomposition directly by applying Zorn’s Lemma.

In [8], Milner observes that,

Nowadays, we should probably say that the infinite case of Dilworth’s decomposition theorem [3] follows from the finite case by “a standard compactness argument.” Depending upon one’s upbringing, what we would have in mind is an application of Gödel’s compactness theorem in logic (cf Church [1]), Tychonoff’s theorem on compact topological spaces, or (if one is a combinatorialist) Rado’s selection lemma [10]. For example, we might argue as follows. Let $\mathcal{P} = (P, \leq)$ be a (partially) ordered set with no antichain of size $k + 1$. Then, by the finite case of Dilworth’s Theorem, any finite subcollection of the set of sentences

$$S = \{ p_{x,1} \vee \cdots \vee p_{x,k} : x \in P \} \cup \{ \neg(p_{x,i} \wedge p_{y,i}) : x \perp y, 1 \leq i \leq k \},$$

has a model (where $x \perp y$ means x and y incomparable in $\mathcal{P}$, and $p_{x,i}$ is a propositional variable with the intended interpretation “ x

belongs to the i^{th} chain"). Hence S has a model and P is the union of the k chains $C_i = \{x \in P : p_{x,i} \text{ is true}\}$ ($1 \leq i \leq k$).

Micha A. Perles' famous counterexample, published in [9], shows that Dilworth's Theorem fails for posets with infinite width. The construction utilizes the Cartesian product of the first uncountable ordinal with itself, denoted as $P = \omega_1 \times \omega_1$, equipped with the standard component-wise product order where $(\alpha, \beta) \leq (\gamma, \delta)$ if and only if $\alpha \leq \gamma$ and $\beta \leq \delta$. An antichain in this poset requires that elements be strictly increasing in one coordinate and strictly decreasing in the other. Because ω_1 is a well-ordered set, it cannot contain an infinite strictly decreasing sequence, which implies that every antichain in P must be finite.

On the other side, Perles's poset cannot be decomposed into a countable collection of chains, thereby shattering the classical finite intuition. By definition, any two elements in a chain $C \subseteq P$ must be comparable, meaning both coordinates must move in the same direction simultaneously. This geometric constraint implies that a chain can intersect any vertical slice $\{\alpha\} \times \omega_1$ at most once, effectively behaving as a partial function from ω_1 to ω_1 . Suppose P could be partitioned into countably many chains $\{C_n\}_{n \in \omega}$, the set of all vertical coordinates covered at a fixed horizontal position α would be a countable union of single points, which is at most countable. However, since the vertical axis ω_1 is inherently uncountable, a vast majority of the grid's surface area remains entirely untouched, proving that a countable chain cover is mathematically impossible.

Concerning previous formal developments on this topic, the only other Mechanization of Dilworth's Theorem in Isabelle/HOL that we are aware of is the recent Archive of Formal Proofs entry *Formal Proof of Dilworth's Theorem* by Maadoori, Meesum, Pillai, Prathamesh and Swami [7], which formalises the *finite* case following Perles's proof and Singh's earlier Coq development [14]. Our work instead addresses the countable-infinite case via propositional compactness.

As previously mentioned, for infinite posets, Dilworth's Theorem requires care: the largest antichain and the smallest (finite) chain decomposition have equal cardinality, whenever there is a finite bound on the size of antichains. In this case, the finite version of Dilworth's Theorem can be combined with a compactness argument based on a graph-coloring reformulation.

Main Contributions. The main contributions of this paper are the following.

- A formal proof of the **countable infinite version of Dilworth's Theorem** obtained from the finite case through a formalization of the k -coloring De Bruijn-Erdős theorem [2] and the propositional compactness theorem.

- As lemmas necessary to obtain the formalization of the infinite version of Dilworth’s Theorem, we have two relevant and nontrivial results:
 - the **formalization of König-Egerváry’s Theorem**, also known as the bipartite matching duality theorem, which states that the size of a maximal matching equals the size of a minimum vertex cover in finite bipartite digraphs. This theorem was formalized using the countable version of Hall’s Theorem for graphs, described in [13], developed as an application of the compactness theorem and Hall’s Theorem for sets [12], and part of the theory on propositional compactness in the Archive of Formal Proofs [15]; and
 - the formalization of the **finite version of Dilworth’s Theorem**, obtained from König-Egerváry’s theorem by associating a bipartite digraph with the partial order and constructing, from a maximum matching and a minimum vertex cover, a smallest chain decomposition and a largest antichain of the same cardinality.

Contents

1	Introduction	1
2	König-Egervary theorem for finite graphs	4
3	Dilworth Finite Theorem	23
4	Dilworth Countable Theorem	54
	References	65

2 König-Egervary theorem for finite graphs

theory *Koenig-Egervary-finite*

This theory formalizes the fact that the Cardinality of a maximal matching equals the Cardinality of a minimum Vertex Cover in finite bipartite digraphs. The formalization uses the countable graph version of Hall Theorem proved in the `Prop_Compactness` as a consequence of the compactness theorem.

Goal: apply the finite version of König-Egervary theorem in the proof of a countable (infinite) version of Dilworth’s theorem.

The countable version of Dilworth’s theorem follows the standard approach

which applies the finite version of Dilworth's theorem, De Bruijn-Erdős k-colouring theorem.

From the finite graph version of Hall theorem follows the finite version of König-Egervary theorem, and then, the finite version of Dilworth's theorem.

Formalization following Jonathan L. Gross' textbook, "Graph Theory and its Applications", 3rd edition, pages 477-478 Hall's theorem implies König-Egervary's theorem

Definition 1 *A vertex cover of a graph G is a set $C \subseteq V(G)$ that contains at least one endpoint of every edge. The vertices in C cover $E(G)$. The next definition is relative to a subset of edges. Then, C such that $\text{vertex_cover } G \ C \ E(G)$ is a vertex cover covering all edges of G .*

definition $\text{vertex-cover}:: ('a, 'b) \text{ pre-digraph} \Rightarrow 'b \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$
where
 $\text{vertex-cover } G \ E \ C \equiv C \subseteq (\text{verts } G) \wedge E \subseteq (\text{arcs } G) \wedge$
 $(\forall e \in E. (\text{head } G \ e) \in C \vee (\text{tail } G \ e) \in C)$

Since no vertex can cover two edges of a matching, the size of every vertex cover is at least the size of every matching.

Lemma 1 *Let M be a matching in a graph G , and let C be a vertex cover of G . Then $\text{card } M \leq \text{card } C$.*

lemma $\text{vertex-in-matching}$:
assumes $\text{dirBD-matching } G \ X \ Y \ E$
shows $\forall e1 \in E. \forall e2 \in E. e1 \neq e2 \longrightarrow$
 $((\text{tail } G \ e1) = v \vee (\text{head } G \ e1) = v) \longrightarrow ((\text{tail } G \ e2) \neq v \wedge (\text{head } G \ e2) \neq v))$
proof(rule+)
show $\bigwedge e1 \ e2.$
 $e1 \in E \implies$
 $e2 \in E \implies$
 $e1 \neq e2 \implies \text{tail } G \ e1 = v \vee \text{head } G \ e1 = v \implies \text{tail } G \ e2 = v \implies \text{False}$
by ($\text{metis assms bipartite-digraph-def dirBD-matching-def}$
 $\text{dir-bipartite-digraph-def disjoint-iff tail-head1}$)
next
show $\bigwedge e1 \ e2.$
 $e1 \in E \implies$
 $e2 \in E \implies e1 \neq e2 \implies \text{tail } G \ e1 = v \vee \text{head } G \ e1 = v \implies \text{head } G \ e2 \neq v$
by ($\text{metis assms bipartite-digraph-def dirBD-matching-def dir-bipartite-digraph-def}$
 $\text{disjoint-iff tail-head1}$)
qed

```

fun f-matching-cover :: ('a,'b) pre-digraph  $\Rightarrow$  'a set  $\Rightarrow$  'b set  $\Rightarrow$  ('a  $\Rightarrow$  'b)
  where
    f-matching-cover G V E = ( $\lambda v$ . (THE e. e  $\in$  E  $\wedge$  (tail G e = v  $\vee$  head G e = v)))

lemma function-f-matching-cover:
  assumes dirBD-matching G X Y E
  and a  $\in$  E and (head G a) = v  $\vee$  (tail G a) = v
  shows f-matching-cover G V E v = a
proof -
  have (THE e. e  $\in$  E  $\wedge$  (tail G e = v  $\vee$  head G e = v)) = a
  proof (rule the-equality)
    show a  $\in$  E  $\wedge$  (tail G a = v  $\vee$  head G a = v) using assms(2,3) by auto
  next
    fix e
    assume hyp1: e  $\in$  E  $\wedge$  (tail G e = v  $\vee$  head G e = v)
    show e = a
    proof (rule ccontr)
      assume hyp2: e  $\neq$  a
      hence (tail G e)  $\neq$  v  $\wedge$  (head G e)  $\neq$  v
      using assms hyp1 hyp2 vertex-in-matching[of G X Y E] by auto
      thus False using hyp1 by auto
    qed
  qed
  thus ?thesis by simp
qed

lemma surj-f-matching-cover1:
  assumes dirBD-matching G X Y E and vertex-cover G E C
  shows  $\forall e \in E. \exists v \in C. \text{f-matching-cover } G \ C \ E \ v = e$ 
proof
  fix e
  assume hyp: e  $\in$  E
  show  $\exists v \in C. \text{f-matching-cover } G \ C \ E \ v = e$ 
  proof -
    have *: (head G e)  $\in$  C  $\vee$  (tail G e)  $\in$  C
      using assms(2) hyp by (unfold vertex-cover-def, auto)
    let ?v1 = (head G e)
    let ?v2 = (tail G e)
    have (?v1 = (head G e)  $\vee$  ?v1 = (tail G e))  $\wedge$  (?v2 = (head G e)  $\vee$  ?v2 = (tail G e))
      by auto
    hence (f-matching-cover G C E ?v1 = e)  $\wedge$  (f-matching-cover G C E ?v2 = e)
      using assms(1) hyp function-f-matching-cover[of G X Y E e - C] by auto
    thus ?thesis using * by auto
  qed
qed

```

lemma *surj-f-matching-cover*:
assumes *dirBD-matching* $G\ X\ Y\ E$ **and** *vertex-cover* $G\ E\ C$
shows $E \subseteq (f\text{-matching-cover } G\ C\ E) \text{ ' } C$
proof
fix e
assume $e \in E$
hence $\exists v \in C. f\text{-matching-cover } G\ C\ E\ v = e$
using *assms surj-f-matching-cover1*[*of* $G\ X\ Y\ E\ C$] **by** *auto*
thus $e \in f\text{-matching-cover } G\ C\ E \text{ ' } C$ **using** *image-def* **by** *auto*
qed

lemma *card-matching-cover*:
assumes *dirBD-matching* $G\ X\ Y\ E$ **and** *vertex-cover* $G\ E\ C$ **and** *finite* C
shows $\text{card } E \leq \text{card } ((f\text{-matching-cover } G\ C\ E) \text{ ' } C)$
proof–
have $E \subseteq (f\text{-matching-cover } G\ C\ E) \text{ ' } C$
using *assms surj-f-matching-cover*[*of* $G\ X\ Y\ E$] **by** *auto*
thus *?thesis* **using** *assms(3)* **by** (*simp add: card-mono*)
qed

lemma *card-matching-cover0*:
assumes *finite* C
shows $\text{card } ((f\text{-matching-cover } G\ C\ E) \text{ ' } C) \leq \text{card } C$ **using**
card-image-le assms **by** *auto*

lemma *card-matching-cover1*:
assumes *dirBD-matching* $G\ X\ Y\ E$ **and** *vertex-cover* $G\ E\ C$ **and** *finite* C
shows $\text{card } E \leq \text{card } C$ **using** *assms card-matching-cover*[*of* $G\ X\ Y\ E\ C$]
card-matching-cover0[*of* $C\ G\ E$] **by** *auto*

Definition 2 *A minimum vertex cover is a vertex cover of minimum size among all vertex cover of the graph.*

definition *minimum-vertex-cover*:: ($'a, 'b$) *pre-digraph* $\Rightarrow 'b\ \text{set} \Rightarrow 'a\ \text{set} \Rightarrow \text{bool}$
where *minimum-vertex-cover* $G\ E\ C$
 $\equiv (\text{vertex-cover } G\ E\ C) \wedge (\forall\ C1. \text{vertex-cover } G\ E\ C1 \longrightarrow \text{card } C \leq \text{card } C1)$

Definition 3 *A maximum matching is a matching of maximum size among all matchings of the graph.*

definition *maximum-dirBD-matching*:: ($'a, 'b$) *pre-digraph* $\Rightarrow 'a\ \text{set} \Rightarrow 'a\ \text{set} \Rightarrow 'b\ \text{set} \Rightarrow \text{bool}$
where *maximum-dirBD-matching* $G\ X\ Y\ E \equiv$
 $\text{dirBD-matching } G\ X\ Y\ E \wedge (\forall\ E1. \text{dirBD-matching } G\ X\ Y\ E1 \longrightarrow \text{card } E1 \leq \text{card } E)$

Let M be a matching in a graph G , and let C be a vertex cover of G such that $|M| = |C|$. Then M is a maximum matching and C is a minimum vertex cover.

lemma *matching-cover-sub1*:

assumes *vertex-cover* G E C **and** $E1 \subseteq E$
shows *vertex-cover* G $E1$ C **using** *assms* **by**(*unfold vertex-cover-def, auto*)

corollary *matching-cover-sub2*:

assumes *vertex-cover* G (*arcs* G) C **and** $E1 \subseteq (\text{arcs } G)$
shows *vertex-cover* G $E1$ C **using** *assms* *matching-cover-sub1* **by** *blast*

lemma *card-matching-cover2*:

assumes *dirBD-matching* G X Y E **and**
vertex-cover G (*arcs* G) C **and**
finite (*verts* G) **and**
 $\text{card } E = \text{card } C$
shows *maximum-dirBD-matching* G X Y $E \wedge$ *minimum-vertex-cover* G (*arcs* G) C

proof

have $C \subseteq (\text{verts } G)$ **using** *assms*(2) **unfolding** *vertex-cover-def* **by** *auto*
hence $*$: *finite* C **using** *assms*(3) *finite-subset*[*of* C *verts* G] **by** *auto*
show *maximum-dirBD-matching* G X Y E
proof(*unfold maximum-dirBD-matching-def, rule conjI*)
show *dirBD-matching* G X Y E **using** *assms*(1) **by** *auto*

next

show $\forall E1. \text{dirBD-matching } G \ X \ Y \ E1 \longrightarrow \text{card } E1 \leq \text{card } E$

proof(*rule allI, rule impI*)

fix $E1$

assume *hyp*: *dirBD-matching* G X Y $E1$

show $\text{card } E1 \leq \text{card } E$

proof(*rule ccontr*)

assume $\neg \text{card } E1 \leq \text{card } E$

hence $\text{card } E < \text{card } E1$ **by** *auto*

hence $\text{card } C < \text{card } E1$ **using** *assms*(4) **by** *auto*

moreover

have $\text{card } E1 \leq \text{card } C$

using *hyp* *dirBD-matching-def*[*of* G X Y $E1$] *matching-cover-sub2*
card-matching-cover1[*of* G X Y $E1$ C] *assms*(2) $*$ **by** *auto*

ultimately

show *False* **by** *auto*

qed

qed

qed

next

show *minimum-vertex-cover* G (*arcs* G) C

proof(*unfold minimum-vertex-cover-def, rule conjI*)

show *vertex-cover* G (*arcs* G) C **using** *assms*(2) **by** *auto*

next

have 2: $E \subseteq \text{arcs } G$ **using** *assms*(1) **unfolding** *dirBD-matching-def* **by** *auto*

```

show  $\forall C1. \text{vertex-cover } G \ (\text{arcs } G) \ C1 \longrightarrow \text{card } C \leq \text{card } C1$ 
proof(rule allI, rule impI)
  fix C1
  assume hyp: vertex-cover G (arcs G) C1
  show card C  $\leq$  card C1
  proof–
    have 1: vertex-cover G E C1
    proof(unfold vertex-cover-def, intro conjI ballI)
      show C1  $\subseteq$  (verts G) using hyp by(unfold vertex-cover-def, auto)
      show E  $\subseteq$  arcs G using 2 by auto
    next
      fix e
      assume e $\in$ E
      hence e $\in$ (arcs G) using 2 by auto
      thus head G e  $\in$  C1  $\vee$  tail G e  $\in$  C1 using hyp by(unfold vertex-cover-def,
auto)
    qed
  next
    show card C  $\leq$  card C1
    proof–
      have 1: vertex-cover G E C1
      proof(unfold vertex-cover-def, intro ballI conjI)
        show C1  $\subseteq$  (verts G) using hyp by(unfold vertex-cover-def, auto)
        show E  $\subseteq$  arcs G using 2 by auto
      fix e
      assume e $\in$ E
      hence e $\in$ (arcs G) using 2 by auto
      thus head G e  $\in$  C1  $\vee$  tail G e  $\in$  C1 using hyp by(unfold vertex-cover-def,
auto)
    qed
    have C1  $\subseteq$  verts G using 1 by(unfold vertex-cover-def, auto)
    hence finite C1 using assms(3) by (simp add: finite-subset)
    hence card E  $\leq$  card C1 using assms(1) 1 card-matching-cover1[of G X
Y E C1] by auto
    thus card C  $\leq$  card C1 using assms(4) by auto
  qed
qed
qed
qed
qed

```

Remark: The converse of the above property does not hold in general; however, it does hold for bipartite graphs.

Theorem 1 (König-Egervary) *If G is a bipartite graph, then the maximum size of a matching in G equals the minimum size of a vertex cover G .*

lemma vertex-subset-BD-decomposition:

assumes *bipartite-digraph* $G\ X\ Y$ **and** $C \subseteq (\text{verts } G)$
shows $C = (C \cap X) \cup (C \cap Y)$
by (*metis* *assms*(1,2) *bipartite-digraph-def* *inf.order-iff* *inf-sup-distrib1*)

lemma *cover-descomposition*:
assumes *bipartite-digraph* $G\ X\ Y$ **and** *vertex-cover* $G\ (\text{arcs } G)\ C$
shows $C = (C \cap X) \cup (C \cap Y)$
using *vertex-subset-BD-decomposition*
by (*metis* *assms*(1,2) *vertex-cover-def*)

lemma *cover-descomposition1*:
assumes *bipartite-digraph* $G\ X\ Y$ **and** *minimum-vertex-cover* $G\ (\text{arcs } G)\ C$
shows $C = (C \cap X) \cup (C \cap Y)$
using *assms*(1,2) *cover-descomposition* *minimum-vertex-cover-def* **by** *blast*

lemma *card-neighbourhood*:
assumes *finite* $(\bigcup (\text{neighbourhood } G'W))$ **and**
 $\text{card } (\bigcup (\text{neighbourhood } G'(W-\{w\}))) = \text{card } (\bigcup (\text{neighbourhood } G'W))$
shows $(\bigcup (\text{neighbourhood } G'(W-\{w\}))) = (\bigcup (\text{neighbourhood } G'W))$ **using**
assms
proof–
have $(\bigcup (\text{neighbourhood } G'(W-\{w\}))) \subseteq (\bigcup (\text{neighbourhood } G'W))$ **using**
assms **by** *auto*
thus *?thesis* **using** *assms* *card-subset-eq*[$\bigcup (\text{neighbourhood } G'W) (\bigcup (\text{neighbourhood } G'(W-\{w\})))$]
by *auto*
qed

lemma *neighbourhood-set*:
assumes *finite* W **and** *finite* $(\bigcup (\text{neighbourhood } G'W))$ **and**
 $(\text{card } W) > \text{card } (\bigcup (\text{neighbourhood } G'W))$
shows $\exists w \in W. (\bigcup (\text{neighbourhood } G'(W-\{w\}))) = (\bigcup (\text{neighbourhood } G'W))$

proof–
 $\{$
fix W
have *finite* $W \implies \text{finite } (\bigcup (\text{neighbourhood } G'W)) \implies (\text{card } W) > \text{card } (\bigcup (\text{neighbourhood } G'W)) \implies$
 $\exists w \in W. (\bigcup (\text{neighbourhood } G'(W-\{w\}))) = (\bigcup (\text{neighbourhood } G'W))$
proof(*induct arbitrary: G rule: finite-psubset-induct*)
case (*psubset* W)
show *?case*
proof (*cases*)
assume $W = \{\}$ **then show** *?thesis*
using *psubset.prem*(2) **by** *auto*
next
assume $W \neq \{\}$
hence $\exists w. w \in W$ **by** *auto*
then obtain w **where** $w: w \in W$ **by** *auto*

```

    have 1: card (W - {w}) = (card W) - 1 using w by auto
    have (⋃ (neighbourhood G (W - {w}))) ⊆ (⋃ (neighbourhood G (W))) by
auto
    hence 2: card (⋃ (neighbourhood G (W - {w}))) ≤ card (⋃ (neighbourhood
G (W)))
        using psubset.prem(1) by (metis card-mono)
    hence card (⋃ (neighbourhood G (W - {w}))) ≤ card (W - {w})
        using psubset.prem(2) using w by auto
    hence (card (⋃ (neighbourhood G (W - {w}))) = card (W - {w})) ∨
        card (⋃ (neighbourhood G (W - {w}))) < card (W - {w}) by auto
    thus ?thesis
proof
    assume hyp: card (⋃ (neighbourhood G (W - {w}))) = card (W - {w})

    have card (W - {w}) ≤ card (⋃ (neighbourhood G (W))) using 2 hyp by
auto
    hence card (⋃ (neighbourhood G (W))) = card (W - {w}) using psub-
set.prem(2) 1
    by auto
    hence card (⋃ (neighbourhood G (W - {w}))) = card (⋃ (neighbourhood
G (W)))
        using hyp by auto
    hence (⋃ (neighbourhood G (W - {w}))) = (⋃ (neighbourhood G (W)))
using card-neighbourhood
    by (metis psubset.prem(1))
    thus ?thesis using w by blast
next
    let ?B = W - {w}
    assume card (⋃ (neighbourhood G (?B))) < card ?B
    hence ∃ z ∈ ?B. (⋃ (neighbourhood G (?B - {z}))) = (⋃ (neighbourhood G
(?B)))
        using w psubset(2) psubset.prem(1)
        by (meson Diff-subset (⋃ (neighbourhood G (W - {w})) ⊆
⋃ (neighbourhood G (W)) card-Diff1-less card-psubset finite-subset
psubset.hyps(1))
    then obtain z where z1: z ∈ ?B and z2: (⋃ (neighbourhood G (?B - {z})))
= (⋃ (neighbourhood G (?B)))
        by auto
    have 1: W - {z} = (?B - {z}) ∪ {w} using w z1 by auto
    have 2: W = ?B ∪ {w} using w by auto
    have (⋃ (neighbourhood G (W - {z}))) = (⋃ (neighbourhood G (W)))
using z2 1 2 by auto
    thus ?thesis using z1 w by auto
qed
qed
qed
}
thus ?thesis using assms by auto
qed

```

definition *induced-subgraph*:: ('a,'b) pre-digraph $\Rightarrow$ 'a set $\Rightarrow$ ('a,'b) pre-digraph

where

induced-subgraph $G \ V \equiv$
 (| *verts* = V ,
arcs = $\{e \mid e. e \in (\text{arcs } G) \wedge (\text{tail } G \ e) \in V \wedge (\text{head } G \ e) \in V\}$,
tail = $(\lambda \ e. \text{tail } G \ e)$,
head = $(\lambda \ e. \text{head } G \ e)$
 |)

lemma *induced-Bipartite-Digraph*:

assumes *bipartite-digraph* $G \ X \ Y$ **and** $X1 \subseteq X$ **and** $Y1 \subseteq Y$

shows *bipartite-digraph* (*induced-subgraph* $G \ (X1 \cup Y1)$) $X1 \ Y1$

proof–

have $X1 \cup Y1 = \text{verts} \ (\text{induced-subgraph } G \ (X1 \cup Y1))$

by (*simp add: induced-subgraph-def*)

moreover

have $X1 \cap Y1 = \{\}$ **using** *assms bipartite-digraph-def*[*of* $G \ X \ Y$] **by** *auto*

moreover

have $(\forall e \in \text{arcs} \ (\text{induced-subgraph } G \ (X1 \cup Y1)).$

$(\text{tail} \ (\text{induced-subgraph } G \ (X1 \cup Y1)) \ e \in X1) =$

$(\text{head} \ (\text{induced-subgraph } G \ (X1 \cup Y1)) \ e \in Y1))$

using *assms*

by (*smt (verit, del-insts)*)

IntI Un-iff bipartite-digraph-def empty-iff induced-subgraph-def mem-Collect-eq

select-convs(2)

select-convs(3) *select-convs*(4) *subset-eq*)

ultimately

show *?thesis using bipartite-digraph-def by blast*

qed

lemma *perfect-matching-induced-subgraph*:

fixes $G :: ('a::\text{countable}, 'b::\text{countable}) \text{pre-digraph}$

assumes *dir-bipartite-digraph* (*induced-subgraph* $G \ V$) $X \ Y$

and $\forall i \in X. \text{finite} \ (\text{neighbourhood} \ (\text{induced-subgraph } G \ V) \ i)$

and $\forall W \subseteq X. \text{card } W \leq \text{card} \ (\bigcup (\text{neighbourhood} \ (\text{induced-subgraph } G \ V) \ `W))$

shows $(\exists E. \text{dirBD-perfect-matching} \ (\text{induced-subgraph } G \ V) \ X \ Y \ E)$

by (*simp add: assms(1) assms(2) assms(3) marriage-sufficiency-graph*)

lemma *vertex-cover-refining*:

assumes *dir-bipartite-digraph* $G \ X \ Y$

and *finite* (*verts* G)

and *vertex-cover* $G \ (\text{arcs } G) \ C$

and $W \subseteq C \cap X$

and $\text{card}(W) > \text{card}(\bigcup (\text{neighbourhood} \ (\text{induced-subgraph } G \ ((C \cap X) \cup (Y - (C \cap Y)))) \ `W))$

shows *vertex-cover* $G \ (\text{arcs } G)$

$((C \cap X - W) \cup ((C \cap Y) \cup \bigcup (\text{neighbourhood } (\text{induced-subgraph } G ((C \cap X) \cup (Y - (C \cap Y)))) 'W)))$
proof(*unfold vertex-cover-def, rule conjI*)
show $C \cap X - W \cup (C \cap Y \cup \bigcup (\text{neighbourhood } (\text{induced-subgraph } G (C \cap X \cup (Y - C \cap Y))) 'W)) \subseteq \text{verts } G$
by(*smt (verit, del-insts) Int-Diff UN-iff Un-iff assms(1,3) bipartite-digraph-def dir-bipartite-digraph-def induced-subgraph-def le-infI1*
le-sup-iff mem-Collect-eq neighbour-def neighbourhood-def_simps(2,3,4) subsetI tail-head vertex-cover-def)
next
show $\text{arcs } G \subseteq \text{arcs } G \wedge (\forall e \in \text{arcs } G. \text{head } G e \in C \cap X - W \cup (C \cap Y \cup \bigcup (\text{neighbourhood } (\text{induced-subgraph } G (C \cap X \cup (Y - C \cap Y))) 'W)) \vee \text{tail } G e \in C \cap X - W \cup (C \cap Y \cup \bigcup (\text{neighbourhood } (\text{induced-subgraph } G (C \cap X \cup (Y - C \cap Y))) 'W)))$
proof(*intro conjI ballI*)
show $\text{arcs } G \subseteq \text{arcs } G$ **by** *auto*
next
let $?Cx = C \cap X$
let $?Cy = C \cap Y$
let $?Cpx = ?Cx - W$
let $?G1 = (\text{induced-subgraph } G (?Cx \cup (Y - ?Cy)))$
let $?NbhGpW = \bigcup (\text{neighbourhood } ?G1 'W)$
fix e
assume $e \in \text{arcs } G$
show $\text{head } G e \in ?Cpx \cup (?Cy \cup ?NbhGpW) \vee \text{tail } G e \in ?Cpx \cup (?Cy \cup ?NbhGpW)$
by(*smt (verit, del-insts) Diff-iff Int-iff UN-iff Un-iff ‹e ∈ arcs G› assms(1,3) induced-subgraph-def*
mem-Collect-eq neighbour-def neighbourhood-def_simps(2,3,4) tail-head vertex-cover-def)
qed
qed

lemma *vertex-cover-refining1*:

assumes *dir-bipartite-digraph* G X Y **and** *finite* ($\text{verts } G$) **and** *vertex-cover* G ($\text{arcs } G$) C
and $W \subseteq C \cap X$
and $\text{card}(W) > \text{card}(\bigcup (\text{neighbourhood } (\text{induced-subgraph } G ((C \cap X) \cup (Y - (C \cap Y)))) 'W))$
shows $\text{card}((C \cap X) - W \cup ((C \cap Y) \cup \bigcup (\text{neighbourhood } (\text{induced-subgraph } G ((C \cap X) \cup (Y - (C \cap Y)))) 'W))) < (\text{card } C)$
proof–
let $?Cx = C \cap X$
let $?Cy = C \cap Y$
let $?Cpx = ?Cx - W$
let $?G1 = (\text{induced-subgraph } G (?Cx \cup (Y - ?Cy)))$

```

let ?NbhGpW =  $\bigcup$  (neighbourhood ?G1'W)
let ?Cpy = ?Cy  $\cup$  ?NbhGpW
have 1: ?Cx  $\cap$  ?Cy = {}
  by (metis assms(1) bipartite-digraph-def dir-bipartite-digraph-def inf-assoc inf-bot-right
inf-left-commute)
have 2: C = ?Cx  $\cup$  ?Cy
  using assms(1,3) cover-descomposition dir-bipartite-digraph-def by blast
have card ?Cpx < card ?Cx - card ?NbhGpW
by (metis (full-types) assms(2,3,4,5) card-Diff-subset card-mono diff-less-mono2

finite-Int infinite-super less-le-trans vertex-cover-def)
hence card ?Cpx + card ?Cy < card ?Cx + card ?Cy - card ?NbhGpW
  by linarith
hence card ?Cpx + card ?Cy + card ?NbhGpW < card ?Cx + card ?Cy by
auto
hence card ?Cpx + card ?Cy + card ?NbhGpW < card C
  by (metis 1 2 assms(2,3) card-Un-disjoint finite-Int rev-finite-subset vertex-cover-def)

thus card (?Cpx  $\cup$  (?Cy  $\cup$  ?NbhGpW)) < card C
  by (smt (verit, ccfv-SIG) add-le-cancel-left card-Un-le linorder-not-less nat-arith.add1
order-trans)
qed

```

corollary *vc-refinable-not-minimal*:

```

assumes dir-bipartite-digraph G X Y
  and finite (verts G)
  and vertex-cover G (arcs G) C
  and W  $\subseteq$  C  $\cap$  X
  and card(W) > card( $\bigcup$  (neighbourhood (induced-subgraph G ((C  $\cap$  X)  $\cup$  (Y
- (C  $\cap$  Y))))'W ))
  shows  $\neg$  minimum-vertex-cover G (arcs G) C
proof
  assume minimum-vertex-cover G (arcs G) C
  show False using vertex-cover-refining vertex-cover-refining1
    by (metis <minimum-vertex-cover G (arcs G) C> assms(1,2,4,5) leD mini-
mum-vertex-cover-def)
qed

```

theorem *Konig-Egervary1*:

```

fixes G :: ('a::countable, 'b::countable) pre-digraph
assumes maximum-dirBD-matching G X Y E and minimum-vertex-cover G
(arcs G) C
  and finite (verts G) and finite C and dir-bipartite-digraph G X Y
  and  $\forall i \in X. \text{finite (neighbourhood G } i)$ 
  and dir-bipartite-digraph (induced-subgraph G ((C  $\cap$  X)  $\cup$  (Y - (C  $\cap$  Y)))) (C
 $\cap$  X) (Y - (C  $\cap$  Y))
  shows ( $\exists E. \text{dirBD-perfect-matching(induced-subgraph G (C } \cap \text{ X } \cup \text{ (Y - C } \cap \text{ Y)) (C } \cap \text{ X) (Y - C } \cap \text{ Y) E)$ )
proof -

```

```

let ?C1 = C ∩ X
let ?C2 = C ∩ Y
let ?G1 = induced-subgraph G (?C1 ∪ (Y − ?C2))
have 1: ?C1 ⊆ X and 2: Y − ?C2 ⊆ Y by auto
have 3: bipartite-digraph G X Y using assms(5) dir-bipartite-digraph-def by
auto
hence 4: bipartite-digraph ?G1 ?C1 (Y − ?C2)
  using 1 2 induced-Bipartite-Digraph[of G X Y ?C1 Y − ?C2] by auto
have *: vertex-cover G (arcs G) C using assms(2)
  by (simp add: minimum-vertex-cover-def)
have **: ∀ W. W ⊆ ?C1 ⟶ (card W) ≤ card (⋃ (neighbourhood ?G1 ‘ W))
proof(rule allI, rule impI)
  fix W::'a set
  assume hyp1: W ⊆ ?C1
  show card W ≤ card (⋃ (neighbourhood (induced-subgraph G (C ∩ X ∪ (Y −
C ∩ Y))) ‘ W))
  proof−
    have 6: finite W
    using finite-subset hyp1 assms(4) by blast
    have 7: (∀ W. ∀ w. W ⊆ ?C1 ∧ w ∈ W ⟶ (neighbourhood ?G1 w) ⊆
(neighbourhood G w))
    proof((rule allI)+, rule impI)
      fix W fix w
      assume hyp: W ⊆ ?C1 ∧ w ∈ W
      show neighbourhood ?G1 w ⊆ neighbourhood G w
      proof
        fix x
        assume x ∈ (neighbourhood ?G1 w)
        hence neighbour ?G1 x w using neighbourhood-def by force
        hence neighbour G x w using 1 2 4 hyp induced-subgraph-def
        by (smt (z3) mem-Collect-eq neighbour-def select-convs(2) select-convs(3)
select-convs(4))
        thus x ∈ (neighbourhood G w) using neighbourhood-def by force
      qed
    qed
  have ∀ W. W ⊆ ?C1 ⟶ (⋃ (neighbourhood ?G1 ‘ W)) ⊆ (⋃ (neighbourhood
G ‘ W))
  proof(intro allI impI)
    fix W
    assume hyp: W ⊆ ?C1
    show (⋃ (neighbourhood ?G1 ‘ W)) ⊆ (⋃ (neighbourhood G ‘ W))
    proof
      fix x
      assume x ∈ (⋃ (neighbourhood ?G1 ‘ W))
      hence ∃ w ∈ W. x ∈ (neighbourhood ?G1 w) by auto
      then obtain w where w: w ∈ W and x ∈ (neighbourhood ?G1 w) by
auto
      hence x ∈ (neighbourhood G w) using hyp 7 by auto
      thus x ∈ (⋃ (neighbourhood G ‘ W)) using w by auto
    end
  end
end

```

```

    qed
  qed
  have  $\forall W. W \subseteq ?C1 \longrightarrow$ 
     $\text{card } W \leq (\text{card } (\bigcup (\text{neighbourhood } ?G1 \text{ ' } W)))$ 
  proof (intro allI impI)
    fix W
    assume hyp1:  $W \subseteq ?C1$ 
    hence hyp2: finite W using 7
    using assms(4) finite-subset by fastforce
    show  $\text{card } W \leq (\text{card } (\bigcup (\text{neighbourhood } ?G1 \text{ ' } W)))$ 
    proof(rule ccontr)
      assume
        hyp:  $\neg \text{card } W \leq (\text{card } (\bigcup (\text{neighbourhood } ?G1 \text{ ' } W)))$ 
      show False using vc-refinable-not-minimal
        by (metis * assms(2,3,5) hyp hyp1 linorder-le-less-linear)
    qed
  qed
  qed
  thus ?thesis
    using hyp1 by blast
  qed
  qed
  thus  $(\exists E. \text{dirBD-perfect-matching } (\text{induced-subgraph } G (C \cap X \cup (Y - C \cap Y))) (C \cap X) (Y - C \cap Y) E)$ 
    using marriage-sufficiency-graph ** assms(7)
    by (smt (verit, del-insts) 4 Int-Un-eq(3) Un-Int-assoc-eq Un-Int-eq(3) assms(4)

    bipartite-digraph-def card.empty card.infinite card-mono
    card-subset-eq empty-iff finite-UN finite-Un order-antisym)
  qed

lemma bd-induced-subgraph:
  assumes bipartite-digraph G X Y and X1  $\subseteq$  X and Y1  $\subseteq$  Y
  shows bipartite-digraph (induced-subgraph (inverse-digraph G) (X1  $\cup$  Y1)) Y1 X1
  by (metis assms(1,2,3) induced-Bipartite-Digraph inverse-bipartite-digraph sup commute)

lemma minimum-vertex-cover-inverse:
  assumes minimum-vertex-cover G (arcs G) C
  shows minimum-vertex-cover (inverse-digraph G) (arcs G) C
  by (smt (verit, ccfv-threshold) assms inverse-digraph-def
    minimum-vertex-cover-def select-convs(1,2,3,4)
    vertex-cover-def)

lemma vc-inv-digraph-is-vc:
  assumes vertex-cover G (arcs G) C
  shows vertex-cover (inverse-digraph G) (arcs G) C
  by (metis assms inverse-digraph-def select-convs(1,2,3,4) vertex-cover-def)

corollary vertex-cover-refining2a:

```

```

assumes dir-bipartite-digraph (inverse-digraph G) Y X
and finite (verts G)
and vertex-cover (inverse-digraph G) (arcs G) C
and  $W \subseteq C \cap Y$ 
and  $\text{card } W > \text{card } (\bigcup (\text{neighbourhood } (\text{induced-subgraph } (\text{inverse-digraph } G) (C \cap Y \cup (X - C \cap X)))) \text{ ' } W))$ 
shows  $\neg \text{minimum-vertex-cover } (\text{inverse-digraph } G) (\text{arcs } G) C$ 
proof
assume minimum-vertex-cover (inverse-digraph G) (arcs G) C
show False
by (metis  $\langle \text{minimum-vertex-cover } (\text{inverse-digraph } G) (\text{arcs } G) C \rangle \text{ assms}(1,2,4,5)$ 
inverse-digraph-def minimum-vertex-cover-def
select-convs(1) simps(2) vc-refinable-not-minimal)
qed

```

theorem *Konig-Egervary2*:

```

fixes G :: ('a::countable, 'b::countable) pre-digraph
assumes maximum-dirBD-matching G X Y E
and minimum-vertex-cover (inverse-digraph G) (arcs G) C
and finite (verts(inverse-digraph G))
and finite C
and dir-bipartite-digraph G X Y
and  $\forall i \in Y. \text{finite } (\text{neighbourhood } (\text{inverse-digraph } G) i)$ 
and dir-bipartite-digraph (induced-subgraph (inverse-digraph G)  $((C \cap Y) \cup (X - (C \cap X)))$ ) (C  $\cap Y$ ) ( $X - (C \cap X)$ )
and  $\text{verts } G \subseteq X \cup Y$ 
shows  $(\exists E. \text{dirBD-perfect-matching } (\text{induced-subgraph } (\text{inverse-digraph } G) (C \cap Y \cup (X - C \cap X))) (C \cap Y) (X - C \cap X) E)$ 
proof–
let ?C1 = C  $\cap X$ 
let ?C2 = C  $\cap Y$ 
let ?G1 = induced-subgraph (inverse-digraph G) (?C2  $\cup (X - ?C1)$ )
have 1: ?C2  $\subseteq Y$  and 2:  $X - ?C1 \subseteq X$  by auto
have 3: bipartite-digraph (inverse-digraph G) Y X
using assms(5) dir-bipartite-digraph-def by (metis inverse-bipartite-digraph)
hence 4: bipartite-digraph ?G1 ?C2 (X - ?C1)
using 1 2 bd-induced-subgraph[of G X Y ?C2 X - ?C1]
by(meson induced-Bipartite-Digraph inverse-bipartite-digraph)
have *: vertex-cover (inverse-digraph G) (arcs (inverse-digraph G)) C using
assms(2)
by (simp add: inverse-digraph-def minimum-vertex-cover-def)
have **:  $\forall W. W \subseteq ?C2 \longrightarrow (\text{card } W) \leq \text{card } (\bigcup (\text{neighbourhood } ?G1 \text{ ' } W))$ 
proof(intro allI impI)
fix W::'a set
assume hyp1:  $W \subseteq ?C2$ 
show  $\text{card } W \leq \text{card } (\bigcup (\text{neighbourhood } (\text{induced-subgraph } (\text{inverse-digraph } G) (C \cap Y \cup (X - C \cap X))) \text{ ' } W))$ 
proof–
have 6: finite W using finite-subset hyp1 assms(4) by blast

```

```

      have  $\gamma$ :  $(\forall W. \forall w. W \subseteq ?C2 \wedge w \in W \longrightarrow (\text{neighbourhood } ?G1 \ w) \subseteq$ 
 $(\text{neighbourhood } (\text{inverse-digraph } G) \ w))$ 
    proof(intro allI impI)
      fix W fix w
      assume hyp:  $W \subseteq ?C2 \wedge w \in W$ 
      show  $(\text{neighbourhood } ?G1 \ w) \subseteq (\text{neighbourhood } (\text{inverse-digraph } G) \ w)$ 
    proof
      fix x
      assume  $x \in (\text{neighbourhood } ?G1 \ w)$ 
      hence neighbour  $?G1 \ x \ w$  using neighbourhood-def by force
      hence neighbour  $(\text{inverse-digraph } G) \ x \ w$  using 1 2 4 hyp induced-subgraph-def
      by (smt (z3) mem-Collect-eq neighbour-def select-convs(2) select-convs(3)
select-convs(4))
      thus  $x \in (\text{neighbourhood } (\text{inverse-digraph } G) \ w)$  using neighbourhood-def
    by force
    qed
  qed
  have  $\forall W. W \subseteq ?C2 \longrightarrow (\bigcup (\text{neighbourhood } ?G1 \ ' W)) \subseteq (\bigcup (\text{neighbourhood } (\text{inverse-digraph } G) \ ' W))$ 
    proof(intro allI impI)
      fix W
      assume hyp:  $W \subseteq ?C2$ 
      show  $(\bigcup (\text{neighbourhood } ?G1 \ ' W)) \subseteq (\bigcup (\text{neighbourhood } (\text{inverse-digraph } G) \ ' W))$ 
    proof
      fix x
      assume  $x \in (\bigcup (\text{neighbourhood } ?G1 \ ' W))$ 
      hence  $\exists w \in W. x \in (\text{neighbourhood } ?G1 \ w)$  by auto
      then obtain w where  $w: w \in W$  and  $x \in (\text{neighbourhood } ?G1 \ w)$  by
auto
      hence  $x \in (\text{neighbourhood } (\text{inverse-digraph } G) \ w)$  using hyp  $\gamma$  by auto
      thus  $x \in (\bigcup (\text{neighbourhood } (\text{inverse-digraph } G) \ ' W))$  using w by auto
    qed
  qed
  hence  $\delta$ :  $\forall W. W \subseteq ?C2 \longrightarrow \text{finite } (\bigcup (\text{neighbourhood } ?G1 \ ' W))$  using
assms(6)
  by (smt (verit, ccfv-threshold) 1 Int-Diff-Un assms(4) finite-UN finite-Un
rev-finite-subset subset-iff)
  have  $\forall W. W \subseteq ?C2 \longrightarrow \text{card } W \leq (\text{card } (\bigcup (\text{neighbourhood } ?G1 \ ' W)))$ 
    proof(intro allI impI)
      fix W
      assume hyp1:  $W \subseteq ?C2$ 
      hence hyp2: finite W using  $\gamma$  using assms(4) finite-subset by fastforce
      show  $\text{card } W \leq (\text{card } (\bigcup (\text{neighbourhood } ?G1 \ ' W)))$ 
    proof(rule ccontr)
      assume hyp:  $\neg \text{card } W \leq (\text{card } (\bigcup (\text{neighbourhood } ?G1 \ ' W)))$  show
False
      by(metis (no-types, lifting) 3 assms(2,3,5,8) bipartite-digraph-def hyp
hyp1

```

```

      inf-sup-aci(5) infinite-super leI minimum-vertex-cover-def inv-dirBD-is-dirBD
vertex-cover-refining2a)
    qed
  qed
  thus ?thesis using hyp1 by blast
  qed
  qed
  thus ( $\exists E. \text{dirBD-perfect-matching } (\text{induced-subgraph } (\text{inverse-digraph } G) (C \cap Y \cup (X - C \cap X)))$ 
    ( $C \cap Y$ ) ( $X - C \cap X$ )  $E$ )
    using marriage-sufficiency-graph[of ( $\text{inverse-digraph } G$ ) ?C2  $X - ?C1$ ] **
  assms(7)
  by (smt (verit, del-insts) Diff-cancel Int-Diff-Un Un-Int-assoc-eq Un-Int-eq(3))
  assms(4)
    card.empty card.infinite card-mono card-subset-eq
    empty-iff finite-UN inf-le2 nle-le perfect-matching-induced-subgraph rev-finite-subset)
  qed

lemma union-dirBd-matching:
  assumes dir-bipartite-digraph  $G$   $X$   $Y$  and  $C = (C \cap X) \cup (C \cap Y)$ 
  and dirBD-perfect-matching ( $\text{induced-subgraph } G (C \cap X \cup (Y - C \cap Y))$ )
  ( $C \cap X$ ) ( $Y - C \cap Y$ )  $E1$ 
  and dirBD-perfect-matching ( $\text{induced-subgraph } (\text{inverse-digraph } G) (C \cap Y \cup (X - C \cap X))$ )
  ( $C \cap Y$ ) ( $X - C \cap X$ )  $E2$ 
  shows dirBD-matching  $G$   $X$   $Y$  ( $E1 \cup E2$ )
proof-
  have 1:  $E1 \cup E2 \subseteq \text{arcs } G$ 
  by (smt (verit, ccfv-threshold) Un-iff assms(3,4) dirBD-matching-def
    dirBD-perfect-matching-def in-mono induced-subgraph-def inverse-digraph-def

    mem-Collect-eq_simps(2) subsetI)
  have 2: ( $\forall e1 \in E1 \cup E2. \forall e2 \in E1 \cup E2. e1 \neq e2 \longrightarrow \text{head } G \ e1 \neq \text{head } G \ e2$ 
 $\wedge \text{tail } G \ e1 \neq \text{tail } G \ e2$ )
  proof
    fix e1
    assume hyp1:  $e1 \in E1 \cup E2$ 
    show ( $\forall e2 \in E1 \cup E2. e1 \neq e2 \longrightarrow \text{head } G \ e1 \neq \text{head } G \ e2 \wedge \text{tail } G \ e1 \neq \text{tail } G \ e2$ )
  proof (intro ballI impI)
    fix e2
    assume hyp2:  $e2 \in E1 \cup E2$ 
    assume hyp3:  $e1 \neq e2$ 
    show  $\text{head } G \ e1 \neq \text{head } G \ e2 \wedge \text{tail } G \ e1 \neq \text{tail } G \ e2$ 
  proof-
    have ( $e1 \in E1 \wedge e2 \in E1$ )  $\vee$  ( $e1 \in E2 \wedge e2 \in E1$ )  $\vee$  ( $e1 \in E1 \wedge e2 \in E2$ )  $\vee$  ( $e1 \in E2 \wedge e2 \in E2$ )
    using hyp1 hyp2 by auto
    thus ?thesis

```

by (smt (verit, ccfv-SIG) DiffE assms(3,4) dirBD-perfect-matching-def
 hyp3 induced-subgraph-def
 inverse-digraph-def select-convs(3,4) tail-head1 vertex-in-matching)
 qed
 qed
 qed
 show ?thesis using assms(1) 1 2 using dirBD-matching-def by blast
 qed

lemma card-prop-induced-dirBD-perfect-matching:
 assumes dirBD-perfect-matching (induced-subgraph G (C $\cap$ X $\cup$ (Y - C $\cap$ Y))) (C $\cap$ X) (Y - C $\cap$ Y) E
 and finite (arcs G)
 shows card (C $\cap$ X) = card E
proof -
 have E $\subseteq$ (arcs G) using assms induced-subgraph-def
 by (metis (no-types, lifting) dirBD-matching-def dirBD-perfect-matching-def
 mem-Collect-eq select-convs(2) subset-iff)
 hence finite E using induced-subgraph-def assms(2)
 using rev-finite-subset by blast
 thus card (C $\cap$ X) = card E using card-perfect-match-in-finitedirBD
 using assms(1) by blast
 qed

lemma card-prop-induced-inv-dirBD-perfect-matching:
 assumes dirBD-perfect-matching (induced-subgraph (inverse-digraph G) (C $\cap$ Y $\cup$ (X - C $\cap$ X))) (C $\cap$ Y) (X - C $\cap$ X) E
 shows card (C $\cap$ Y) = card E using assms card-perfect-match-in-finitedirBD
 by (smt (verit) bijective-f-perfect-matching card-image dirBD-matching-tail-edge-unicity
 dirBD-perfect-matching-def function-f-perfect-matching inj-on-def tail-head1)

lemma finite-neighbourhood-aux:
 assumes dir-bipartite-digraph G X Y and finite (verts G)
 shows $\forall i \in (\text{verts } G). \text{finite } (\text{neighbourhood } G i)$
 by (smt (verit, del-insts) Un-iff assms(1,2) bipartite-digraph-def dir-bipartite-digraph-def
 mem-Collect-eq neighbour-def neighbourhood-def rev-finite-subset subsetI
 tail-head)

lemma finite-neighbourhood:
 assumes dir-bipartite-digraph G X Y and finite (verts G)
 shows $\forall i \in X. \text{finite } (\text{neighbourhood } G i)$
 using finite-neighbourhood-aux
 by (metis (no-types, lifting) Un-iff assms(1,2) bipartite-digraph-def dir-bipartite-digraph-def)

lemma finite-inverse-neighbourhood:
 assumes dir-bipartite-digraph G X Y and finite (verts G)
 shows $\forall i \in Y. \text{finite } (\text{neighbourhood } (\text{inverse-digraph } G) i)$
proof -

```

have  $Y \subseteq (\text{verts } G)$ 
by (metis Un-iff assms(1) bipartite-digraph-def dir-bipartite-digraph-def subsetI)

thus ?thesis using inverse-bipartite-digraph[of  $G \ X \ Y$ ]
    finite-neighbourhood-aux[of inverse-digraph  $G \ Y \ X$ ] finite-neighbourhood[of
inverse-digraph  $G \ Y \ X$ ]
by (metis assms(1,2) inv-dirBD-is-dirBD inverse-digraph-def select-convs(1))
qed

lemma finite-vertex-cover:
  assumes dir-bipartite-digraph  $G \ X \ Y$  and finite (verts  $G$ ) and vertex-cover  $G$ 
  (arcs  $G$ )  $C$ 
  shows finite  $C$ 
  by (metis assms(2,3) rev-finite-subset vertex-cover-def)

theorem Konig-Egervary:
  fixes  $G :: ('a::countable, 'b::countable) \text{pre-digraph}$ 
  assumes dir-bipartite-digraph  $G \ X \ Y$  and finite (verts  $G$ )
  and maximum-dirBD-matching  $G \ X \ Y \ E$  and minimum-vertex-cover  $G$  (arcs
 $G$ )  $C$ 
  shows card  $E = \text{card } C$ 
proof–
  let ?C1 =  $C \cap X$ 
  let ?C2 =  $C \cap Y$ 
  let ?G0 = induced-subgraph  $G$  (?C1  $\cup$  ( $Y - ?C2$ ))
  let ?G1 = induced-subgraph (inverse-digraph  $G$ ) (?C2  $\cup$  ( $X - ?C1$ ))
  have 1:  $C = (C \cap X) \cup (C \cap Y)$  using cover-descomposition assms(1) dir-bipartite-digraph-def
  assms(3)
  by (metis assms(4) minimum-vertex-cover-def)
  have 2:  $(C \cap X) \cap (C \cap Y) = \{\}$  using assms(1) dir-bipartite-digraph-def
  by (metis bipartite-digraph-def induced-Bipartite-Digraph inf-le2)
  have 3: finite  $C$ 
  using finite-vertex-cover assms(1,2,4) minimum-vertex-cover-def by blast
  have dir-bipartite-digraph ?G0 ?C1 ( $Y - ?C2$ )
  proof–
  have 1: bipartite-digraph ?G0 ?C1 ( $Y - ?C2$ )
  using assms(1) induced-Bipartite-Digraph dir-bipartite-digraph-def by fast-
  force
  have 2: tails ?G0  $\subseteq ?C1$ 
  proof
  fix  $x$ 
  assume hyp:  $x \in \text{tails } ?G0$ 
  show  $x \in ?C1$ 
  proof–
  have  $\exists e. e \in \text{arcs } ?G0 \wedge x = (\text{tail } ?G0 \ e)$  using hyp tails-def
  by (smt (verit, best) mem-Collect-eq)
  thus ?thesis
  by (metis (lifting) 1 IntI Int-Diff-Un Un-iff assms(1)
  bipartite-digraph-def dir-bipartite-digraph-def induced-subgraph-def

```

```

      mem-Collect-eq select-convs(2,3) tail-head)
    qed
  qed
  have 3: (∀ e1 ∈ arcs (induced-subgraph G (?C1 ∪ (Y − ?C2))).
    ∀ e2 ∈ arcs (induced-subgraph G (?C1 ∪ (Y − ?C2))).
    (e1 = e2) =
      (head (induced-subgraph G (?C1 ∪ (Y − ?C2))) e1 = head (induced-subgraph
G (?C1 ∪ (Y − ?C2))) e2 ∧
      tail (induced-subgraph G (?C1 ∪ (Y − ?C2))) e1 = tail (induced-subgraph
G (?C1 ∪ (Y − ?C2))) e2))
    by (metis (no-types, lifting) assms(1) dir-bipartite-digraph-def induced-subgraph-def
mem-Collect-eq
      select-convs(2,3,4))
  show ?thesis using 1 2 3 dir-bipartite-digraph-def by blast
  qed
  have (∃ E1. dirBD-perfect-matching (induced-subgraph G (C ∩ X ∪ (Y − C ∩
Y))) (C ∩ X) (Y − C ∩ Y) E1)
    by (smt (verit, ccfv-threshold) 3 Konig-Egervary1 Un-iff ⟨dir-bipartite-digraph
(induced-subgraph G (C ∩ X ∪ (Y − C ∩ Y))) (C ∩ X) (Y − C ∩ Y)⟩
      assms(1,2,3,4) bipartite-digraph-def dir-bipartite-digraph-def finite-neighbourhood-aux
subset-iff)
  then obtain E1 where E1: dirBD-perfect-matching (induced-subgraph G (C ∩
X ∪ (Y − C ∩ Y))) (C ∩ X) (Y − C ∩ Y) E1 by auto
  have dir-bipartite-digraph ?G1 ?C2 (X − ?C1)
  proof −
    have 1: bipartite-digraph ?G1 ?C2 (X − ?C1)
      by (metis Diff-subset assms(1) bd-induced-subgraph dir-bipartite-digraph-def
inf.cobounded2 inf-sup-aci(5))
    have 2: tails ?G1 ⊆ ?C2
  proof
    fix x
    assume hyp: x ∈ tails ?G1
    show x ∈ C ∩ Y
  proof −
    have ∃ e. e ∈ arcs ?G1 ∧ x = (tail ?G1 e) using hyp tails-def
      by (smt (verit, best) mem-Collect-eq)
    thus ?thesis
      by (metis (no-types, lifting) 2 Int-iff Un-Diff-Int Un-iff assms(1) bipar-
tite-digraph-def dir-bipartite-digraph-def
        induced-subgraph-def inverse-digraph-def mem-Collect-eq select-convs(2,3)
tail-head)
    qed
  qed
  have 3: (∀ e1 ∈ arcs ?G1. ∀ e2 ∈ arcs ?G1.
    (e1 = e2) = (head ?G1 e1 = head ?G1 e2 ∧ tail ?G1 e1 = tail ?G1 e2))
    by (metis (no-types, lifting) assms(1) dir-bipartite-digraph-def induced-subgraph-def
inverse-digraph-def mem-Collect-eq
      select-convs(2,3,4))
  show ?thesis using 1 2 3 dir-bipartite-digraph-def by blast

```

```

qed
have (∃ E2. dirBD-perfect-matching ?G1 (C ∩ Y) (X - C ∩ X) E2)
  by (smt (verit, best) 3 Konig-Egervary2 <dir-bipartite-digraph (induced-subgraph
(inverse-digraph G) (C ∩ Y ∪ (X - C ∩ X))) (C ∩ Y) (X - C ∩ X)>
    assms(1,2,3,4) bipartite-digraph-def dir-bipartite-digraph-def finite-neighbourhood
inverse-digraph-def minimum-vertex-cover-inverse select-convs(1)
    subset-refl inv-dirBD-is-dirBD)
  then obtain E2 where E2: dirBD-perfect-matching ?G1 (C ∩ Y) (X - C ∩
X) E2
    by auto
  have 4: dirBD-matching G X Y (E1 ∪ E2) using union-dirBd-matching assms(1)
1 E1 E2 by blast
  have 5: finite (E1 ∪ E2) using E1 E2
    by (metis 3 4 assms(4) dirBD-matching-def finite-surj
    matching-cover-sub2 minimum-vertex-cover-def surj-f-matching-cover)
  have 6: card (C ∩ X) = card E1 using E1 card-prop-induced-dirBD-perfect-matching
5 card-perfect-match-in-finitedirBD by blast
  have 7: card (C ∩ Y) = card E2 using E2 card-prop-induced-inv-dirBD-perfect-matching
by blast
  have 8: card C = card E1 + card E2
    by (metis 1 2 3 6 7 card-Un-disjoint finite-Int)
  have E1 ∩ E2 = {}
  proof(rule ccontr)
    assume E1 ∩ E2 ≠ {}
    hence ∃ e. e ∈ E1 ∧ e ∈ E2 by auto
    then obtain e where e: e ∈ E1 ∧ e ∈ E2 by auto
    hence tail G e ∈ (C ∩ X) ∧ tail G e ∈ (C ∩ Y) using E1 E2 using
dirBD-perfect-matching-def
    by (metis (no-types, lifting) DiffE induced-subgraph-def inverse-digraph-def
select-convs(3)
   _simps(4) tail-head1)
    hence 1: tail G e ∈ X ∩ Y by auto
    hence X ∩ Y ≠ {} by auto
  have X ∩ Y = {} using assms(1) bipartite-digraph-def by (metis dir-bipartite-digraph-def)

  thus False using 1 by auto
qed
hence 9: card (E1 ∪ E2) = card C
  using 5 8 card-Un-disjoint by auto
hence maximum-dirBD-matching G X Y (E1 ∪ E2) using card-matching-cover2
  using 4 assms(2,4) minimum-vertex-cover-def by blast
thus ?thesis
  by (metis 9 assms(3) maximum-dirBD-matching-def
    order-antisym)
qed

```

3 Dilworth Finite Theorem

In a finite graph, the Cardinality of the smallest chain decomposition and the largest anti-chain are equal.

theory *Dilworth-Finite*

imports

Main

HOL-Library.Disjoint-Sets

Koenig-Egervary-finite

begin

locale *part-order* =

fixes $A :: 'a \text{ set}$

fixes $r :: 'a \text{ rel}$

assumes $p\text{-}o\text{-translation} : \text{partial-order-on } A \ r$

locale *part-order-countable* = *part-order* $A \ r$

for $A :: 'a::\text{countable set}$ **and** $r :: 'a \text{ rel}$

definition *chain*:: $'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow \text{bool}$

where $\text{chain } B \ A \ r \equiv B \subseteq A \wedge \text{partial-order-on } A \ r \wedge \text{total-on } B \ r$

definition *chain-decomposition*:: $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ set set} \Rightarrow \text{bool}$

where $\text{chain-decomposition } A \ r \ C \equiv \text{partition-on } A \ C \wedge (\forall B \in C. (\text{chain } B \ A \ r))$

lemma *chain-decomposition* $\{\} \ r \ \{\}$

by (*simp add: chain-decomposition-def partition-on-empty*)

definition *smallest-chain-decomposition*:: $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ set set} \Rightarrow \text{bool}$

where $\text{smallest-chain-decomposition } A \ r \ C \equiv$
 $(\text{chain-decomposition } A \ r \ C) \wedge (\forall P. \text{chain-decomposition } A \ r \ P \longrightarrow \text{card } C \leq \text{card } P)$

definition *anti-total* :: $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow \text{bool}$

where $\text{anti-total } A \ r \equiv (\forall x \in A. \forall y \in A. x \neq y \longrightarrow (x, y) \notin r \wedge (y, x) \notin r)$

definition *anti-chain*:: $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$

where $\text{anti-chain } A \ r \ B \equiv B \subseteq A \wedge \text{partial-order-on } A \ r \wedge \text{anti-total } B \ r$

definition *largest-antichain*:: $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$

where $\text{largest-antichain } A \ r \ B \equiv \text{anti-chain } A \ r \ B \wedge (\forall C. \text{anti-chain } A \ r \ C \longrightarrow \text{card } C \leq \text{card } B)$

definition *relation-to-digraph*:: $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow (('a + 'a), ('a + 'a) \times ('a + 'a))\text{pre-digraph}$

where

$\text{relation-to-digraph } A \ r \equiv$

```

(| verts = A <+> A,
 arcs = {(Inl x, Inr y) | x y. x ∈ A ∧ y ∈ A ∧ (x,y) ∈ r ∧ x ≠ y},
 tail = λ (x,y). x,
 head = λ (x,y). y
|)

```

lemma *rel-to-digraph-is-dir-bipartite*:

```

assumes r ⊆ A × A
shows dir-bipartite-digraph (relation-to-digraph A r) (Inl ‘ A) (Inr ‘ A)
proof(unfold dir-bipartite-digraph-def, rule conjI )
show bipartite-digraph (relation-to-digraph A r) (Inl ‘ A) (Inr ‘ A)
proof(unfold bipartite-digraph-def, intro conjI)
show Inl ‘ A ∪ Inr ‘ A = verts (relation-to-digraph A r)
  by (unfold relation-to-digraph-def, auto)
show Inl ‘ A ∩ Inr ‘ A = {}
  by auto
show (∀ e ∈ arcs (relation-to-digraph A r).
  (tail (relation-to-digraph A r) e ∈ Inl ‘ A) =
  (head (relation-to-digraph A r) e ∈ Inr ‘ A))
proof
  fix e
  assume Hip: e ∈ arcs (relation-to-digraph A r)
  show (tail (relation-to-digraph A r) e ∈ Inl ‘ A) =
    (head (relation-to-digraph A r) e ∈ Inr ‘ A)
  proof–
    from Hip obtain x y where e: e = (Inl x, Inr y) ∧ (x,y) ∈ r
    by(unfold relation-to-digraph-def, auto)
    hence Hip1: e = ((Inl x), (Inr y)) ∧ (x,y) ∈ r
    by auto
    hence Hip2: (Inl x) = tail (relation-to-digraph A r) e
      ∧ (Inr y) = head (relation-to-digraph A r) e ∧ x ∈ A ∧ y ∈ A

    using assms by(unfold relation-to-digraph-def, auto)
    hence x ∈ A ∧ y ∈ A by auto
    hence (Inl x) ∈ Inl ‘ A ∧ (Inr y) ∈ Inr ‘ A
    by auto
    hence (tail (relation-to-digraph A r) e ∈ Inl ‘ A) ∧
      (head (relation-to-digraph A r) e ∈ Inr ‘ A)
    using Hip1 e by(unfold relation-to-digraph-def, auto)
    thus (tail (relation-to-digraph A r) e ∈ Inl ‘ A) =
      (head (relation-to-digraph A r) e ∈ Inr ‘ A)
    by auto
  qed
qed
qed
next
show tails (relation-to-digraph A r) ⊆ Inl ‘ A ∧
  (∀ e1 ∈ arcs (relation-to-digraph A r).
    ∀ e2 ∈ arcs (relation-to-digraph A r).

```

$(e1 = e2) =$
 $(\text{head } (\text{relation-to-digraph } A \ r) \ e1 = \text{head } (\text{relation-to-digraph } A \ r) \ e2 \wedge$
 $\text{tail } (\text{relation-to-digraph } A \ r) \ e1 = \text{tail } (\text{relation-to-digraph } A \ r) \ e2))$
by (smt (z3) image-eqI mem-Collect-eq prod.simps(2) relation-to-digraph-def
select-convs(2,3,4) subsetI tails-def)
qed

lemma *exists-min-vertex-cover*:
assumes *dir-bipartite-digraph* $G \ X \ Y$ **and** *finite* $(X \cup Y)$
shows $\exists C. \text{minimum-vertex-cover } G \ (\text{arcs } G) \ C$
proof –
let $?P = \{C. \text{vertex-cover } G \ (\text{arcs } G) \ C\}$
have *finite* $(\text{verts } G)$
by (metis assms(1,2) bipartite-digraph-def dir-bipartite-digraph-def)
hence 1: *finite* $?P$ **using** assms *finite-Pow-iff* **by** (simp add: vertex-cover-def)
have *vertex-cover* $G \ (\text{arcs } G) \ (\text{verts } G)$ **using** assms(1) *bipartite-digraph-def*
vertex-cover-def
by (smt (verit, del-insts) Un-iff dir-bipartite-digraph-def mem-Collect-eq sub-
set-iff tails-def)
hence 2: $?P \neq \{\}$ **by** auto
have $\exists C \in ?P. (\forall C' \in ?P. \text{card } C \leq \text{card } C')$ **using** 1 2
by (metis vertex-cover $G \ (\text{arcs } G) \ (\text{verts } G)$ ex-has-least-nat mem-Collect-eq)
thus *?thesis* **using** *minimum-vertex-cover-def* **by** auto
qed

definition *roots* :: $'a \text{ set} \Rightarrow (('a + 'a) \times ('a + 'a)) \text{ set} \Rightarrow 'a \text{ set}$ **where**
 $\text{roots } A \ M \equiv \{x \in A. \forall y. (\text{Inl } y, \text{Inr } x) \notin M\}$

lemma *not-root-has-pred*:
assumes $x \in A$ **and** $x \notin \text{roots } A \ M$
shows $\exists y. (\text{Inl } y, \text{Inr } x) \in M$
proof –
from assms(2)
have $\neg (\forall y. (\text{Inl } y, \text{Inr } x) \notin M)$
unfolding *roots-def* **using** assms(1) **by** auto
then obtain y **where** $(\text{Inl } y, \text{Inr } x) \in M$ **by** auto
thus *?thesis* **by** blast
qed

definition *match-rel* ::
 $(('a + 'a) \times ('a + 'a)) \text{ set} \Rightarrow ('a \times 'a) \text{ set}$
where
 $\text{match-rel } M =$
 $\{(x, y). (\text{Inl } x, \text{Inr } y) \in M\}$

lemma *matching-edges-strict*:
assumes *dirBD-matching* $(\text{relation-to-digraph } A \ r) \ (\text{Inl } 'A) \ (\text{Inr } 'A) \ M$ **and**
 $(x, y) \in \text{match-rel } M$
shows $x \in A \wedge y \in A \wedge (x, y) \in r \wedge x \neq y$

proof –
 have $(Inl\ x, Inr\ y) \in M$ **using** $assms(2)$ $match\text{-}rel\text{-}def[of\ M]$ **by** *auto*
moreover
 have $M \subseteq arcs\ (relation\text{-}to\text{-}digraph\ A\ r)$ **using** $assms(1)$ $dirBD\text{-}matching\text{-}def$
by *metis*
ultimately
 have $(Inl\ x, Inr\ y) \in arcs\ (relation\text{-}to\text{-}digraph\ A\ r)$ **using** $assms(1)$ $relation\text{-}to\text{-}digraph\text{-}def$
by *auto*
 thus *?thesis*
by $(simp\ add: relation\text{-}to\text{-}digraph\text{-}def)$
qed

definition *chain-from-root* **where**
 $chain\text{-}from\text{-}root\ a\ A\ M =$
 $\{x \in A. (a, x) \in (match\text{-}rel\ M)^*\}$

lemma *successor-unique*:
 assumes $(x, y1) \in match\text{-}rel\ M$ **and** $(x, y2) \in match\text{-}rel\ M$
 and $dirBD\text{-}matching\ (relation\text{-}to\text{-}digraph\ A\ r)\ (Inl\ 'A)\ (Inr\ 'A)\ M$
 shows $y1 = y2$
proof(*rule ccontr*)
 assume *hip*: $y1 \neq y2$
 have $1: (Inl\ x, Inr\ y1) \in M$ **using** $assms(1,2)$ $match\text{-}rel\text{-}def$
by $(simp\ add: match\text{-}rel\text{-}def)$
 have $2: (Inl\ x, Inr\ y2) \in M$ **using** $assms(1,2)$ $match\text{-}rel\text{-}def$
by $(simp\ add: match\text{-}rel\text{-}def)$
 have $(Inl\ x, Inr\ y1) \neq (Inl\ x, Inr\ y2)$ **using** *hip* **by** *auto*
 hence $x \neq x$ **using** $dirBD\text{-}matching\text{-}def\ assms(3)$
by $(metis\ (no\text{-}types,\ lifting)\ 1\ 2\ relation\text{-}to\text{-}digraph\text{-}def\ select\text{-}convs(3)\ split\text{-}conv)$
 thus *False* **by** *auto*
qed

lemma *predecessor-unique*:
 assumes $(x1, y) \in match\text{-}rel\ M$ **and** $(x2, y) \in match\text{-}rel\ M$
 and $dirBD\text{-}matching\ (relation\text{-}to\text{-}digraph\ A\ r)\ (Inl\ 'A)\ (Inr\ 'A)\ M$
 shows $x1 = x2$
proof(*rule ccontr*)
 assume *hip*: $x1 \neq x2$
 have $1: (Inl\ x1, Inr\ y) \in M$ **using** $assms(1,2)$ $match\text{-}rel\text{-}def$
by $(simp\ add: match\text{-}rel\text{-}def)$
 have $2: (Inl\ x2, Inr\ y) \in M$ **using** $assms(1,2)$ $match\text{-}rel\text{-}def$
by $(simp\ add: match\text{-}rel\text{-}def)$
 have $(Inl\ x1, Inr\ y) \neq (Inl\ x2, Inr\ y)$ **using** *hip* **by** *auto*
 hence $y \neq y$ **using** $dirBD\text{-}matching\text{-}def\ assms(3)$
by $(metis\ (no\text{-}types,\ lifting)\ 2\ Product\text{-}Type.\ Collect\text{-}case\text{-}prodD\ assms(2)\ fst\text{-}conv\ match\text{-}rel\text{-}def\ prod.\ simps(2)\ relation\text{-}to\text{-}digraph\text{-}def\ select\text{-}convs(4)\ snd\text{-}conv)$
 thus *False* **by** *auto*

qed

lemma *root-unique*:

assumes

ax: $(a, x) \in (\text{match-rel } M)^{\wedge*}$ **and**

bx: $(b, x) \in (\text{match-rel } M)^{\wedge*}$ **and**

ar: $a \in \text{roots } A \ M$ **and**

br: $b \in \text{roots } A \ M$ **and**

mt: *dirBD-matching* (*relation-to-digraph* $A \ r$) (*Inl* ‘ A) (*Inr* ‘ A) M

shows $a = b$

using *ax bx ar br*

proof (*induction arbitrary*; *b rule: rtrancl-induct*)

case *base*

then show *?case*

by (*metis* (*lifting*) *CollectD Product-Type.Collect-case-prodD match-rel-def roots-def*

rtranclE split-pairs2)

next

case (*step y z*)

have *bz*: $(b, z) \in (\text{match-rel } M)^{\wedge*}$

using *step.prem*s **by** *auto*

show *?case*

proof (*cases b = z*)

case *True*

hence $(y, z) \in \text{match-rel } M$

using *step.hyps* **by** *auto*

hence $(y, b) \in \text{match-rel } M$

using *True* **by** *auto*

hence $\exists u. (u, b) \in \text{match-rel } M$

by *auto*

thus *?thesis*

using *step.prem*s **unfolding** *roots-def*

by (*simp add: match-rel-def*)

next

case *False*

then obtain *w* **where**

$(b, w) \in (\text{match-rel } M)^{\wedge*}$

$(w, z) \in \text{match-rel } M$

using *bz* **by** (*metis rtranclE*)

moreover

have $(y, z) \in \text{match-rel } M$

using *step.hyps* **by** *auto*

ultimately have $y = w$

by (*meson mt predecessor-unique*)

thus *?thesis*

using *step.IH step.prem*s

using $\langle (b, w) \in (\text{match-rel } M)^{\wedge*} \rangle$ **by** *blast*

qed

qed

definition *pred-rel*:: $((a + 'a) \times (a + 'a)) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ('a \times 'a) \text{ set}$ **where**
pred-rel $M A = \{(x,y). (Inl\ x, Inr\ y) \in M \wedge x \in A \wedge y \in A\}$

lemma *pred-rel-subset-strict*:

assumes *dirBD-matching* (*relation-to-digraph* $A\ r$) (*Inl* ' A) (*Inr* ' A) M
shows *pred-rel* $M A \subseteq r - Id$

proof

fix p

assume $p \in \text{pred-rel } M A$

then obtain $x\ y$ **where**

p-def: $p = (x,y)$

and *edge*: $(Inl\ x, Inr\ y) \in M$

unfolding *pred-rel-def*

by *auto*

have $(x,y) \in r \wedge x \neq y$ **using** *matching-edges-strict*[*of* $A\ r\ M\ x\ y$]

by (*simp add: assms edge match-rel-def*)

thus $p \in r - Id$

using *p-def* **by** *auto*

qed

lemma (*in part-order*)*strict-order-wf*:

assumes *finite* A

shows *wf* ($r - Id$)

proof–

have *fin-r* : *finite* r **using** *p-o-translation partial-order-on-def*[*of* $A\ r$] *assms*

finite-cartesian-product[*of* $A\ A$] *finite-subset*[*of* $r\ A \times A$] *preorder-on-def*[*of* $A\ r$]

by (*simp add: refl-on-def*)

thus *?thesis* **using** *p-o-translation partial-order-on-well-order-on*[*of* $r\ A$] **by** *auto*

qed

lemma (*in part-order*)*finite-pred-wf*:

assumes *finite* A **and**

dirBD-matching (*relation-to-digraph* $A\ r$)(*Inl* ' A) (*Inr* ' A) M

shows *wf* (*pred-rel* $M A$)

proof –

have *pred-rel* $M A \subseteq r - Id$

using *assms*(2) *pred-rel-subset-strict* **by** *blast*

moreover

have *wf* ($r - Id$)

using *strict-order-wf* *assms*(1,2)

by *auto*

ultimately

show *?thesis*

using *wf-subset*

by *blast*

qed

```

lemma (in part-order) exists-root-reaching:
  assumes finite A
    and dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M
    and  $x \in A$ 
  shows  $\exists a \in \text{roots } A \ M. (a, x) \in (\text{match-rel } M)^{\wedge*}$ 
proof -
  have wf-rel: wf (pred-rel M A)
    using finite-pred-wf assms
    by blast
  from assms(3)
  show ?thesis
proof (induction x rule: wf-induct[OF wf-rel])
  case (1 x)
  have xA:  $x \in A$ 
    using 1 by blast
  show ?case
proof (cases  $x \in \text{roots } A \ M$ )
  case True
  have  $(x, x) \in (\text{match-rel } M)^{\wedge*}$ 
    by simp
  thus ?thesis
    using True by blast
next
  case False
  from not-root-has-pred[OF xA False]
  obtain y where edge:  $(\text{Inl } y, \text{Inr } x) \in M$ 
    by blast
  have yA:  $y \in A$ 
    using match-rel-def matching-edges-strict assms(2) edge by fastforce
  have  $(y, x) \in \text{pred-rel } M \ A$ 
    using pred-rel-def edge xA yA
    by auto
  from 1
  obtain a where
     $a \in \text{roots } A \ M$  and  $(a, y) \in (\text{match-rel } M)^{\wedge*}$ 
    using  $\langle (y, x) \in \text{pred-rel } M \ A \rangle yA$  by blast
  then have  $(a, x) \in (\text{match-rel } M)^{\wedge*}$ 
    using edge
    by (simp add: match-rel-def rtrancl.rtrancl-into-rtrancl)
  thus ?thesis
    using  $\langle a \in \text{roots } A \ M \rangle$ 
    by blast
qed
qed
qed

```

```

lemma single-valued-rft-trnstv-closure :
  assumes single-valued r

```

```

and    $(a,x) \in r^*$ 
and    $(a,y) \in r^*$ 
shows  $x = y \vee (x,y) \in r^* \vee (y,x) \in r^*$ 
using assms single-valued-confluent[of r a x y] by auto

lemma functional-rtrancl-linear:
assumes
  func:  $\bigwedge x y1 y2. (x,y1) \in r \implies (x,y2) \in r \implies y1 = y2$ 
and ax:  $(a,x) \in r^*$ 
and ay:  $(a,y) \in r^*$ 
shows  $x = y \vee (x,y) \in r^* \vee (y,x) \in r^*$ 
using ay
proof (induction rule: rtrancl-induct)
  case base
  then show ?case
    using ax by auto
next
  case (step y z)
  then show ?case
    by (metis converse-rtranclE func rtrancl.rtrancl-into-rtrancl rtrancl.rtrancl-refl)
qed

lemma reachable-linear-from-root:
assumes
   $(a,x) \in (\text{match-rel } M)^*$  and
   $(a,y) \in (\text{match-rel } M)^*$  and
  dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M
shows
   $x = y \vee (x,y) \in (\text{match-rel } M)^* \vee (y,x) \in (\text{match-rel } M)^*$ 
by (meson assms(1,2,3) functional-rtrancl-linear successor-unique)

lemma match-rel-subset-r:
assumes dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M
shows match-rel M  $\subseteq r$ 
proof
  fix p
  assume p  $\in$  match-rel M
  hence  $\exists x.\exists y. p = (x,y) \wedge (x,y) \in \text{match-rel } M$ 
    by (metis surj-pair)
  then obtain x y where p: p = (x,y) and (x,y)  $\in$  match-rel M by auto
  hence 1: (Inl x, Inr y)  $\in$  M unfolding match-rel-def by auto
  hence (Inl x, Inr y)  $\in$  arcs (relation-to-digraph A r)
    using assms dirBD-matching-def by fastforce
  hence (x,y)  $\in r$  unfolding relation-to-digraph-def
    by auto
  thus p  $\in r$  using p by auto
qed

```

definition *chain-decomposition-order* :: 'a set $\Rightarrow$ 'a rel $\Rightarrow ((\text{'a} + \text{'a}) \times (\text{'a} + \text{'a}))$
set $\Rightarrow$ 'a set set

where

chain-decomposition-order A r M =
 $\{ \text{chain-from-root } a \ A \ M \mid a. a \in (\text{roots } A \ M) \}$

lemma *match-rel-not-empty*: $a \in A \longrightarrow (a, a) \in (\text{match-rel } M)^\wedge^*$
by *blast*

context *part-order*

begin

lemma *match-rel-rtrancl-in-r*:

assumes

dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M
and $a \in A$ **and** $(a, x) \in (\text{match-rel } M)^\wedge^*$

shows $(a, x) \in r$

using *assms*(3)

proof (*induction rule: rtrancl-induct*)

case *base*

then show ?*case* **using** *assms*(1)

by (*meson* *assms*(2) *p-o-translation partial-order-onD*(1) *reft-onD*)

next

case (*step* y z)

then have $(y, z) \in r$

using *step.prem*s

using *assms*(1) *match-rel-subset-r*

by *blast*

moreover have $(a, y) \in r$

using *step.IH* **by** *auto*

ultimately show ?*case*

using *assms*

by (*metis* *p-o-translation partial-order-onD*(2) *transD*)

qed

lemma *reachable-comparable*:

assumes

$(a, x) \in (\text{match-rel } M)^\wedge^*$ **and**

$(a, y) \in (\text{match-rel } M)^\wedge^*$ **and**

dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M **and**

$a \in A$

shows

$(x, y) \in r \vee (y, x) \in r$

proof –

have *sub*: *match-rel* M $\subseteq$ r

using *match-rel-subset-r* *assms*(4)

using *assms*(3) **by** *blast*

have *axr*: $(a, x) \in r$

by (*meson* *assms*(1,3,4) *part-order.match-rel-rtrancl-in-r* *part-order-axioms*)

have *ayr*: $(a, y) \in r$

```

    by (meson assms(2,3,4) part-order.match-rel-rtrancl-in-r part-order-axioms)
  from axr ayr
  show ?thesis
    by (smt (verit, best) assms(1,2,3,4) match-rel-rtrancl-in-r matching-edges-strict
        reachable-linear-from-root rtrancl.simps)
qed

```

lemma *chain-from-root-is-chain*:

```

  assumes
    finite A and
    dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M and a ∈ A
  shows chain (chain-from-root a A M) A r
proof -
  have subset:
    chain-from-root a A M ⊆ A
    by (simp add: chain-from-root-def)
  moreover
  have total: total-on (chain-from-root a A M) r
  proof
    fix x y
    assume
      x ∈ chain-from-root a A M and y ∈ chain-from-root a A M
    hence (a,x) ∈ (match-rel M)* and (a,y) ∈ (match-rel M)* and x ∈ A and
    y ∈ A
    unfolding chain-from-root-def by auto
    thus (x,y) ∈ r ∨ (y,x) ∈ r
    using reachable-comparable assms by metis
  qed
  ultimately
  show ?thesis
    using chain-def assms
    using p-o-translation by blast
qed

```

lemma *chain-decomposition-cover*:

```

  assumes
    finite A
    dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M
  shows
    ⋃ (chain-decomposition-order A r M) = A
proof
  show ⋃ (chain-decomposition-order A r M) ⊆ A
    unfolding chain-from-root-def
    by (smt (verit, ccfv-SIG) Union-least chain-decomposition-order-def
        chain-from-root-def mem-Collect-eq subsetI)
  next
  show A ⊆ ⋃ (chain-decomposition-order A r M)
  proof
    fix x

```

```

assume  $hip: x \in A$ 
obtain  $a$  where
   $a1: a \in \text{roots } A \ M$  and
   $a2: (a, x) \in (\text{match-rel } M)^*$ 
  using  $\text{exists-root-reaching assms } \langle x \in A \rangle$ 
  by  $\text{metis}$ 
hence  $(a, x) \in (\text{match-rel } M)^*$  using  $a1 \ a2$ 
  by  $\text{blast}$ 
thus  $x \in \bigcup (\text{chain-decomposition-order } A \ r \ M)$ 
  by  $(\text{smt } (\text{verit}, \text{del-ists}) \text{ Union-iff } \langle x \in A \rangle \ a1 \ \text{chain-decomposition-order-def}$ 
     $\text{chain-from-root-def mem-Collect-eq})$ 
qed
qed

lemma  $\text{chain-decomposition-disjoint}$ :
  assumes  $\text{finite } A$  and  $\text{dirBD-matching } (\text{relation-to-digraph } A \ r) (\text{Inl } 'A) (\text{Inr } 'A) \ M$ 
  shows  $\text{disjoint } (\text{chain-decomposition-order } A \ r \ M)$ 
  unfolding  $\text{disjoint-def}$ 
proof  $(\text{intro ballI impI})$ 
  fix  $c1 \ c2$ 
  assume  $c1 \in \text{chain-decomposition-order } A \ r \ M$ 
  and  $c2 \in \text{chain-decomposition-order } A \ r \ M$ 
  and  $c1 \neq c2$ 
  show  $c1 \cap c2 = \{\}$ 
  proof  $(\text{rule ccontr})$ 
    assume  $hip: c1 \cap c2 \neq \{\}$ 
    obtain  $a \ b$  where  $a: c1 = \text{chain-from-root } a \ A \ M \ a \in \text{roots } A \ M$ 
    and  $b: c2 = \text{chain-from-root } b \ A \ M \ b \in \text{roots } A \ M$ 
    by  $(\text{smt } (\text{verit}, \text{ccfv-SIG}) \langle c1 \in \text{chain-decomposition-order } A \ r \ M \rangle$ 
       $\langle c2 \in \text{chain-decomposition-order } A \ r \ M \rangle \ \text{chain-decomposition-order-def}$ 
       $\text{mem-Collect-eq})$ 
    have  $\exists x. (x \in \text{chain-from-root } a \ A \ M) \wedge (x \in \text{chain-from-root } b \ A \ M)$  using
       $hip \ a \ b$  by  $\text{auto}$ 
    then obtain  $x$  where  $x: (x \in \text{chain-from-root } a \ A \ M) \wedge (x \in \text{chain-from-root } b \ A \ M)$  by  $\text{auto}$ 
    hence  $(x \in A \wedge (a, x) \in (\text{match-rel } M)^*) \wedge (x \in A \wedge (b, x) \in (\text{match-rel } M)^*)$ 

    using  $\text{chain-from-root-def}$ 
    by  $(\text{smt } (\text{verit}) \ a(1,2) \ \text{assms}(2) \ b(1,2) \ \text{disjoint-iff hip mem-Collect-eq}$ 
       $\text{part-order.p-o-translation part-order-axioms}$ 
       $\text{root-unique})$ 
    hence  $a=b$ 
    by  $(\text{meson } a(2) \ \text{assms}(2) \ b(2) \ \text{part-order.p-o-translation part-order-axioms}$ 
       $\text{root-unique})$ 
    thus  $\text{False}$ 
    using  $\langle c1 \neq c2 \rangle \ a(1) \ b(1)$  by  $\text{blast}$ 
  qed
qed

```

```

lemma chain-decomposition-non-empty:
  assumes roots  $A \ M \neq \{\}$  and
    dirBD-matching (relation-to-digraph  $A \ r$ ) (Inl '  $A$ ) (Inr '  $A$ )  $M$ 
  shows  $\{\} \notin \text{chain-decomposition-order } A \ r \ M$ 
  using chain-decomposition-order-def chain-from-root-def match-rel-not-empty
  by (smt (verit) equals0D mem-Collect-eq roots-def)

lemma chain-decomposition-partition:
  assumes  $r \subseteq A \times A$  and
    dirBD-matching (relation-to-digraph  $A \ r$ ) (Inl '  $A$ ) (Inr '  $A$ )  $M$ 
    and finite  $A$ 
  shows partition-on  $A$  (chain-decomposition-order  $A \ r \ M$ )
  using partition-on-def[of  $A$  chain-decomposition-order  $A \ r \ M$ ]
proof
  have 1:  $\bigcup (\text{chain-decomposition-order } A \ r \ M) = A$ 
    using assms chain-decomposition-cover by blast
  moreover
  have 2: disjoint (chain-decomposition-order  $A \ r \ M$ )
    using assms chain-decomposition-disjoint by auto
  moreover
  have 3:  $\{\} \notin \text{chain-decomposition-order } A \ r \ M$ 
    using assms chain-decomposition-non-empty
    by (smt (verit, best) chain-decomposition-order-def emptyE mem-Collect-eq)
  ultimately
  show ?thesis using partition-on-def by auto
qed

lemma exist-chain-decomposition:
  assumes  $r \subseteq A \times A$  and
    dirBD-matching (relation-to-digraph  $A \ r$ ) (Inl '  $A$ ) (Inr '  $A$ )  $M$ 
    and finite  $A$ 
  shows chain-decomposition  $A \ r$  (chain-decomposition-order  $A \ r \ M$ )
proof–
  have partition-on  $A$  (chain-decomposition-order  $A \ r \ M$ )
    using assms chain-decomposition-partition by auto
  moreover
  have  $\forall B \in \text{chain-decomposition-order } A \ r \ M. \text{chain } B \ A \ r$ 
  proof
    fix  $B$ 
    assume  $B \in \text{chain-decomposition-order } A \ r \ M$ 
    thus chain  $B \ A \ r$  using assms chain-from-root-is-chain
    by (smt (verit) chain-decomposition-order-def mem-Collect-eq p-o-translation
roots-def)
  qed
  ultimately show ?thesis
    using chain-decomposition-def by blast
qed
end

```

lemma *max-card-in-set*:
fixes $M :: 'a \text{ set set}$
assumes *finite* M **and** $M \neq \{\}$
shows $\exists m \in M. \forall m' \in M. \text{card } m' \leq \text{card } m$
by (*metis* *Max-ge* *assms*(1,2) *finite-imageI* *image-eqI* *obtains-MAX*)

lemma *maximum-matching-digraph*:
assumes *dir-bipartite-digraph* $G \ X \ Y$ **and** *finite* (*arcs* G)
shows $\exists M. \text{maximum-dirBD-matching } G \ X \ Y \ M$
proof –
let $?M = \{M. \text{dirBD-matching } G \ X \ Y \ M\}$
have $a: \forall M \in ?M. \text{finite } M$ **using** *dirBD-matching-def*
by (*metis* *assms*(2) *mem-Collect-eq* *rev-finite-subset*)
have $1: \text{finite } ?M$
proof –
have $?M \subseteq \text{Pow } (\text{arcs } G)$
using *dirBD-matching-def* **by** *fastforce*
thus *finite* $?M$ **by** (*simp* *add: assms*(2) *finite-subset*)
qed
have $2: \{\} \in ?M$
unfolding *dirBD-matching-def* **using** *assms*(1) **by** *fastforce*
have $\exists M \in ?M. (\forall M' \in ?M. \text{card } M' \leq \text{card } M)$ **using** $a \ 1 \ 2$
by (*metis* *equals0D* *max-card-in-set*)
thus *?thesis*
by (*simp* *add: maximum-dirBD-matching-def*)
qed

lemma *chain-from-root-inj*:
assumes
dirBD-matching (*relation-to-digraph* $A \ r$) (*Inl* ' A) (*Inr* ' A) M
shows
inj-on ($\lambda a. \text{chain-from-root } a \ A \ M$) (*roots* $A \ M$)
proof –
have $\bigwedge a \ b.$
 $a \in \text{roots } A \ M \implies$
 $b \in \text{roots } A \ M \implies$
 $\text{chain-from-root } a \ A \ M = \text{chain-from-root } b \ A \ M \implies$
 $a = b$
proof –
fix $a \ b$
assume
 $a \in \text{roots } A \ M$ **and**
 $b \in \text{roots } A \ M$ **and**
 $\text{chain-from-root } a \ A \ M = \text{chain-from-root } b \ A \ M$
then **have** $a \in \text{chain-from-root } b \ A \ M$
unfolding *chain-from-root-def*
by (*metis* (*lifting*) *match-rel-not-empty* *mem-Collect-eq* *roots-def*)
then **have** $(b, a) \in (\text{match-rel } M)^*$

```

    unfolding chain-from-root-def by auto
  moreover
  have  $(a,a) \in (\text{match-rel } M)^\wedge^*$ 
    by auto
  ultimately show  $a = b$ 
    by (metis (lifting) CollectD Product-Type.Collect-case-prodD
         $\langle a \in \text{roots } A \ M \rangle \text{ match-rel-def roots-def rtrancl.cases snd-conv}$ )
qed
thus ?thesis
  unfolding inj-on-def by auto
qed

```

lemma *chain-decomposition-as-image*:

```

chain-decomposition-order  $A \ r \ M = (\lambda a. \text{chain-from-root } a \ A \ M) \text{ ' roots } A \ M$ 
by (simp add: chain-decomposition-order-def setcompr-eq-image)

```

lemma *card-chain-decomposition*:

```

assumes
  dirBD-matching (relation-to-digraph  $A \ r$ ) ( $\text{Inl ' } A$ ) ( $\text{Inr ' } A$ )  $M$  and finite
  (roots  $A \ M$ )
shows
  card (chain-from-root ' roots  $A \ M$ ) = card (roots  $A \ M$ )
by (metis (no-types, lifting) assms(1) card-image chain-from-root-inj inj-on-def)

```

```

fun f-match-rel ::  $('a \times 'a) \Rightarrow (('a + 'a) \times ('a + 'a))$ 
where
  f-match-rel  $(x,y) = (\text{Inl } x, \text{Inr } y)$ 

```

lemma *inj-match-rel*:

```

assumes dirBD-matching (relation-to-digraph  $A \ r$ ) ( $\text{Inl ' } A$ ) ( $\text{Inr ' } A$ )  $M$ 
shows inj f-match-rel
by (unfold inj-def, auto)

```

lemma *surj-match-rel*:

```

assumes dirBD-matching (relation-to-digraph  $A \ r$ ) ( $\text{Inl ' } A$ ) ( $\text{Inr ' } A$ )  $M$ 
shows  $(\forall z.\forall w. (z,w) \in M \longrightarrow (\exists x.\exists y. (x,y) \in \text{match-rel } M \wedge (\text{Inl } x, \text{Inr } y) = (z,w)))$ 
proof (intro allI impI)
  fix  $z \ w$ 
  assume hip:  $(z, w) \in M$ 
  show  $\exists x \ y. (x, y) \in \text{match-rel } M \wedge (\text{Inl } x, \text{Inr } y) = (z, w)$ 
  proof -
    have  $(\exists x.\exists y. (x,y) \in r \wedge x \in A \wedge y \in A \wedge x \neq y \wedge (\text{Inl } x, \text{Inr } y) = (z,w))$ 
      using hip
    by (smt (verit, del-insts) assms dirBD-matching-def mem-Collect-eq
        relation-to-digraph-def select-convs(2) subset-iff)
    then obtain  $x \ y$  where  $(x,y) \in r \wedge x \in A \wedge y \in A \wedge x \neq y \wedge (\text{Inl } x, \text{Inr } y) = (z,w)$ 
      by auto
    hence  $(\text{Inl } x, \text{Inr } y) \in M \wedge (\text{Inl } x, \text{Inr } y) = (z,w)$  using hip match-rel-def

```

using *hip* by *fastforce*
 hence $(x, y) \in \text{match-rel } M \wedge (\text{Inl } x, \text{Inr } y) = (z, w)$ using *match-rel-def* by
auto
 thus ?thesis by *blast*
 qed
 qed

lemma *surj-match-rel1*:

assumes *dirBD-matching* (*relation-to-digraph* A r) ($\text{Inl} \text{ ` } A$ ) ( $\text{Inr} \text{ ` } A$) M
 shows $f\text{-match-rel} \text{ ` } \text{match-rel } M = M$
proof
 show $f\text{-match-rel} \text{ ` } \text{match-rel } M \subseteq M$
proof
 fix X
 assume $X \in f\text{-match-rel} \text{ ` } \text{match-rel } M$
 hence $\exists z. \exists w. X = (\text{Inl } z, \text{Inr } w) \wedge (\text{Inl } z, \text{Inr } w) \in f\text{-match-rel} \text{ ` } \text{match-rel } M$
 using *image-iff* by *fastforce*
 then obtain z w where *: $X = (\text{Inl } z, \text{Inr } w) \wedge (\text{Inl } z, \text{Inr } w) \in f\text{-match-rel}$
 $\text{ ` } \text{match-rel } M$ by *auto*
 hence $(z, w) \in \text{match-rel } M$ by *fastforce*
 hence $(\text{Inl } z, \text{Inr } w) \in M$
 by (*simp add: match-rel-def*)
 thus $X \in M$ using * by *auto*
 qed
next
 show $M \subseteq f\text{-match-rel} \text{ ` } \text{match-rel } M$
proof
 fix X
 assume $X \in M$
 hence $\exists z. \exists w. X = (\text{Inl } z, \text{Inr } w) \wedge (\text{Inl } z, \text{Inr } w) \in M$ using *assms*
 by (*metis f-match-rel.cases surj-match-rel*)
 then obtain z w where $X = (\text{Inl } z, \text{Inr } w) \wedge (\text{Inl } z, \text{Inr } w) \in M$ by *auto*
 hence $(z, w) \in \text{match-rel } M \wedge X = (\text{Inl } z, \text{Inr } w) \wedge (\text{Inl } z, \text{Inr } w) \in M$
 by (*simp add: match-rel-def*)
 thus $X \in f\text{-match-rel} \text{ ` } \text{match-rel } M$
 by *force*
 qed
 qed

lemma *card-match-rel-eq*:

assumes *dirBD-matching* (*relation-to-digraph* A r) ($\text{Inl} \text{ ` } A$ ) ( $\text{Inr} \text{ ` } A$) M and
finite M
 and *finite* ($\text{match-rel } M$)
 shows $\text{card } (\text{match-rel } M) = \text{card } M$
proof –
 have *inj f-match-rel*
 using *assms*(1) *inj-match-rel* by *blast*
 moreover
 have $f\text{-match-rel} \text{ ` } \text{match-rel } M = M$

using *surj-match-rel1* *assms* **by** *blast*
ultimately show *?thesis*
by (*metis card-image inj-on-subset subset-UNIV*)
qed

definition *has-pred*:: *'a set $\Rightarrow$ (('a + 'a) $\times$ ('a + 'a)) set $\Rightarrow$ 'a set* **where**
has-pred A M = {x $\in$ A. ($\exists$ y. (y,x) $\in$ match-rel M)}

lemma *roots-partition*:
roots A M = A - has-pred A M
unfolding *roots-def has-pred-def*
by (*smt (z3) Collect-cong DiffD2 DiffI Diff-subset case-prodE case-prodI f-match-rel.simps*
match-rel-def mem-Collect-eq minus-set-def subset-eq)

lemma *disjoint-roots-pred*:
roots A M $\cap$ has-pred A M = {}
unfolding *roots-def has-pred-def*
by (*simp add: disjoint-iff match-rel-def*)

lemma *union-roots-pred*:
roots A M $\cup$ has-pred A M = A
using *roots-partition disjoint-roots-pred*
by (*simp add: Collect-conj-eq has-pred-def roots-partition*)

lemma *card-partition*:
assumes *finite A*
shows *card A = card (roots A M) + card (has-pred A M)*
using *disjoint-roots-pred union-roots-pred*
by (*metis assms card-Un-disjoint disjoint-roots-pred finite-Un union-roots-pred*)

lemma *has-pred-image*:
assumes *dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M*
shows *has-pred A M = snd ' (match-rel M)*
unfolding *has-pred-def*
by (*smt (verit, best) Collect-cong Range.RangeI RangeE assms(1) image-def*
matching-edges-strict mem-Collect-eq
snd-eq-Range)

lemma *inj-snd-match-rel*:
assumes
dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M
shows *inj-on snd (match-rel M)*
using *assms*
unfolding *dirBD-matching-def match-rel-def inj-on-def*
by (*metis (lifting) Product-Type.Collect-case-prodD assms*
predecessor-unique split-pairs2 sum.inject(1,2) surj-match-rel)

lemma *card-has-pred*:

assumes
dirBD-matching (*relation-to-digraph* A r) ($Inl \text{ ' } A$) ($Inr \text{ ' } A$) M **and** *finite*
(*match-rel* M)
shows $card\ (has-pred\ A\ M) = card\ (match-rel\ M)$
using *assms* *has-pred-image inj-snd-match-rel card-image* **by** *metis*

lemma *card-roots*:

assumes
dirBD-matching (*relation-to-digraph* A r) ($Inl \text{ ' } A$) ($Inr \text{ ' } A$) M **and** *finite* A
and *finite* M **and** *finite* (*match-rel* M)
shows
 $card\ (roots\ A\ M) = card\ A - card\ M$
proof –
have $card\ A = card\ (roots\ A\ M) + card\ (has-pred\ A\ M)$
using *card-partition assms* **by** *auto*
moreover
have $card\ (has-pred\ A\ M) = card\ (match-rel\ M)$
using *card-has-pred assms* **by** *blast*
moreover
have $card\ (match-rel\ M) = card\ M$
using *card-match-rel-eq assms* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed

lemma *finite-arcs*:

assumes *finite* A **and** $r \subseteq A \times A$
shows *finite* (*arcs* (*relation-to-digraph* A r))
proof–
have *finite* r
using *assms(1,2) finite-subset* **by** *blast*
thus *?thesis* **using** *relation-to-digraph-def[of A r]*
by (*smt* (*verit, ccfv-SIG*) *InlI InrI SigmaI assms(1) finite-Plus-iff finite-cartesian-product*
mem-Collect-eq rev-finite-subset select-convs(2) subrelI)
qed

lemma *finite-verts*:

assumes *finite* A
shows *finite* (*verts* (*relation-to-digraph* A r))
using *relation-to-digraph-def assms*
by (*smt* (*verit*) *finite-Plus-iff select-convs(1)*)

lemma *finite-matching*:

assumes
 $r \subseteq A \times A$ **and**
dirBD-matching (*relation-to-digraph* A r) ($Inl \text{ ' } A$) ($Inr \text{ ' } A$) M **and** *finite* A
shows *finite* M **using** *assms dirBD-matching-def finite-arcs*
by (*metis finite-subset*)

lemma (**in** *part-order*)*chain-decomposition-cardinality*:

```

assumes
  dirBD-matching (relation-to-digraph A r) (Inl ' A) (Inr ' A) M and finite A
shows
  card (chain-decomposition-order A r M) = card A - card M
proof -
  have chain-decomposition-order A r M =
    (λa. chain-from-root a A M) ' roots A M
    using chain-decomposition-as-image by blast
  moreover
  have card (chain-from-root ' roots A M)
    = card (roots A M)
  proof -
    have finite (roots A M)
      using assms
      by (simp add: roots-def)
    moreover
    have inj-on (λa. chain-from-root a A M) (roots A M)
      using chain-from-root-inj assms by blast
    ultimately show ?thesis
      by (simp add: card-image inj-on-def)
  qed
  moreover
  have card (roots A M) = card A - card M
    using card-roots assms finite-matching finite-cartesian-product match-rel-subset-r
    p-o-translation partial-order-onD(4) rev-finite-subset
    by meson
  ultimately show ?thesis
    by (metis assms(1) card-image chain-from-root-inj)
  qed

lemma cover-minimum:
  assumes minimum-vertex-cover G E C and finite C
  shows  $\forall c \in C. \exists e \in E. c = (\text{head } G \ e) \vee c = (\text{tail } G \ e)$ 
proof(rule ccontr)
  assume  $\neg(\forall c \in C. \exists e \in E. c = (\text{head } G \ e) \vee c = (\text{tail } G \ e))$ 
  hence  $\exists c \in C. \forall e \in E. c \neq (\text{head } G \ e) \wedge c \neq (\text{tail } G \ e)$  by auto
  then obtain c where c1:  $c \in C$  and c2:  $\forall e \in E. c \neq (\text{head } G \ e) \wedge c \neq (\text{tail } G \ e)$ 
  by auto
  have *: vertex-cover G E (C - {c})
  proof -
    have C - {c}  $\subseteq$  verts G
      using assms minimum-vertex-cover-def vertex-cover-def
      by (metis c1 insert-Diff insert-subset)
    moreover
    have E  $\subseteq$  arcs G using assms minimum-vertex-cover-def vertex-cover-def
      by metis
    moreover
    have  $(\forall e \in E. \text{head } G \ e \in C - \{c\} \vee \text{tail } G \ e \in C - \{c\})$  using c1 c2
      by (metis assms(1) insert-Diff insert-iff minimum-vertex-cover-def vertex-cover-def)

```

```

ultimately
show ?thesis using vertex-cover-def by blast
qed
have card (C - {c}) < card C using c1 assms(2)
by (metis card-Diff1-less)
thus False using * assms(1) minimum-vertex-cover-def
using leD by blast
qed

context part-order
begin
lemma vertex-cover-reduction:
  assumes (∀ e ∈ arcs (relation-to-digraph A r).
    head (relation-to-digraph A r) e ∈ C ∨
    tail (relation-to-digraph A r) e ∈ C)
  and Inl x ∈ C and Inr x ∈ C
  shows (∀ e ∈ arcs (relation-to-digraph A r).
    head (relation-to-digraph A r) e ∈ C - {Inr x} ∨
    tail (relation-to-digraph A r) e ∈ C - {Inr x}) ∨
    (∀ e ∈ arcs (relation-to-digraph A r).
    head (relation-to-digraph A r) e ∈ C - {Inl x} ∨
    tail (relation-to-digraph A r) e ∈ C - {Inl x})
proof(rule ccontr)
  let ?G = relation-to-digraph A r
  let ?E = arcs ?G
  assume
    ¬((∀ e ∈ ?E.
      head ?G e ∈ C - {Inr x} ∨
      tail ?G e ∈ C - {Inr x}) ∨
      (∀ e ∈ ?E.
        head ?G e ∈ C - {Inl x} ∨
        tail ?G e ∈ C - {Inl x}))
  hence h: ¬(∀ e ∈ ?E.
    head ?G e ∈ C - {Inr x} ∨
    tail ?G e ∈ C - {Inr x}) and
    h0: ¬(∀ e ∈ ?E.
      head ?G e ∈ C - {Inl x} ∨
      tail ?G e ∈ C - {Inl x}) by auto
  from h have ∃ y. ∃ z. ∃ e. e = (Inl y, Inr z) ∧ (head ?G e ∉ C - {Inr x} ∧
    tail ?G e ∉ C - {Inl x})
  by (metis (no-types, lifting) Diff-iff insertI1 old.prod.case relation-to-digraph-def
    select-convs(3,4))
  then obtain y z e where e: e ∈ ?E ∧ e = (Inl y, Inr z) and h: head ?G e ∉ C
    - {Inr x} and
    t: tail ?G e ∉ C - {Inl x}
  by (smt (verit, ccfv-threshold) mem-Collect-eq h relation-to-digraph-def
    select-convs(2))
  hence 1: e ∈ ?E ∧ e = (Inl y, Inr z) ∧ head ?G e = Inr z ∧ tail ?G e = Inl y
  ∧

```

$head\ ?G\ e \notin C - \{Inr\ x\} \wedge tail\ ?G\ e \notin C - \{Inr\ x\}$
using e *relation-to-digraph-def*[of $A\ r$] **by** *auto*
hence $*$: $e = (Inl\ y, Inr\ z) \wedge (Inr\ z) \notin C - \{Inr\ x\} \wedge (Inl\ y) \notin C - \{Inr\ x\}$
using $e\ h\ t$ **by** *auto*
hence a : $(y, z) \in r \wedge y \neq z$ **using** *relation-to-digraph-def*[of $A\ r$]
using e **by** *auto*
have $(Inl\ y) \neq (Inr\ x)$ **by** *auto*
hence b : $(Inl\ y) \notin C$ **using** $*$ **by** *auto*
have $\exists c \in C. c = Inl\ y \vee c = Inr\ z$ **using** $e\ assms(1)\ 1$
by *auto*
then obtain c **where** $c: c \in C \wedge (c = Inl\ y \vee c = Inr\ z)$ **by** *auto*
hence $Inr\ z = Inr\ x$ **using** $*$ **by** *auto*
hence $h1$: $(y, x) \in r \wedge y \neq x$ **using** a **by** *auto*
from $h0$ **have** $\exists y1. \exists z1. \exists e. e = (Inl\ y1, Inr\ z1) \wedge (head\ ?G\ e \notin C - \{Inl\ x\} \wedge tail\ ?G\ e \notin C - \{Inl\ x\})$
by (*smt* (*verit*, *best*) *mem-Collect-eq relation-to-digraph-def select-convs(2)*)
then obtain $y1\ z1\ e$ **where** $e: e \in ?E \wedge e = (Inl\ y1, Inr\ z1)$ **and** h : $head\ ?G\ e \notin C - \{Inl\ x\}$ **and**
 t : $tail\ ?G\ e \notin C - \{Inl\ x\}$
by (*smt* (*verit*, *best*) *CollectD h0 relation-to-digraph-def select-convs(2)*)
hence 2 : $e \in ?E \wedge e = (Inl\ y1, Inr\ z1) \wedge head\ ?G\ e = Inr\ z1 \wedge tail\ ?G\ e = Inl\ y1 \wedge$
 $head\ ?G\ e \notin C - \{Inl\ x\} \wedge tail\ ?G\ e \notin C - \{Inl\ x\}$
using e *relation-to-digraph-def*[of $A\ r$] **by** *auto*
hence $**$: $e = (Inl\ y1, Inr\ z1) \wedge (Inr\ z1) \notin C - \{Inl\ x\} \wedge (Inl\ y1) \notin C - \{Inl\ x\}$
using $e\ h\ t$ **by** *auto*
hence $a1$: $(y1, z1) \in r \wedge y1 \neq z1$ **using** *relation-to-digraph-def*[of $A\ r$]
using e **by** *auto*
have $(Inr\ z1) \neq (Inl\ x)$ **by** *auto*
hence $b1$: $(Inr\ z1) \notin C$ **using** $**$ **by** *auto*
have $\exists c \in C. c = Inl\ y1 \vee c = Inr\ z1$ **using** $2\ assms(1)$ **by** *auto*
then obtain c **where** $c: c \in C \wedge (c = Inl\ y1 \vee c = Inr\ z1)$ **by** *auto*
hence $Inl\ y1 = Inl\ x$ **using** $**$ **by** *auto*
hence $h2$: $(x, z1) \in r \wedge z1 \neq x$ **using** $a1$ **by** *auto*
hence $(y, z1) \in r \wedge y \neq z1$ **using** $h0$
by (*metis antisymD h1 p-o-translation partial-order-onD(2,3) transD*)
hence $(Inl\ y, Inr\ z1) \in ?E$
by (*smt* (*verit*) $1\ e\ mem-Collect-eq\ prod.inject\ relation-to-digraph-def\ select-convs(2)\ sum.inject(1,2)$)
thus *False* **using** $b\ b1\ assms(1)$
by (*metis* (*no-types*, *lifting*) *prod.simps(2) relation-to-digraph-def select-convs(3,4)*)
qed

lemma *vertex-cover-subset*:

assumes *vertex-cover* (*relation-to-digraph* $A\ r$) (*arcs* (*relation-to-digraph* $A\ r$))
 C
and $Inl\ x \in C \wedge Inr\ x \in C$
shows *vertex-cover* (*relation-to-digraph* $A\ r$) (*arcs* (*relation-to-digraph* $A\ r$))

```

( $C - \{Inl\ x\}$ )
   $\vee$  vertex-cover (relation-to-digraph  $A\ r$ ) (arcs (relation-to-digraph  $A\ r$ ))
( $C - \{Inr\ x\}$ )
proof(rule disjE)
  let  $?G = \text{relation-to-digraph } A\ r$ 
  let  $?E = \text{arcs } ?G$ 
  show  $(\forall e \in ?E.$ 
     $\text{head } ?G\ e \in C - \{Inr\ x\} \vee$ 
     $\text{tail } ?G\ e \in C - \{Inr\ x\}) \vee$ 
     $(\forall e \in ?E.$ 
     $\text{head } ?G\ e \in C - \{Inl\ x\} \vee$ 
     $\text{tail } ?G\ e \in C - \{Inl\ x\})$  using assms vertex-cover-def vertex-cover-reduction
  by (metis (no-types, lifting))
next
  let  $?G = \text{relation-to-digraph } A\ r$ 
  let  $?E = \text{arcs } ?G$ 
  assume  $h: (\forall e \in ?E. \text{head } ?G\ e \in C - \{Inr\ x\} \vee \text{tail } ?G\ e \in C - \{Inr\ x\})$ 
  have vertex-cover (relation-to-digraph  $A\ r$ ) (arcs (relation-to-digraph  $A\ r$ )) ( $C - \{Inr\ x\}$ )
  proof–
    have  $C - \{Inr\ x\} \subseteq \text{verts } (\text{relation-to-digraph } A\ r)$ 
    using assms(1) vertex-cover-def
    by (metis assms(2) insert-Diff insert-subset)
    thus  $?thesis$  using  $h$  vertex-cover-def
    by blast
  qed
  thus  $?thesis$  by auto
next
  let  $?G = \text{relation-to-digraph } A\ r$ 
  let  $?E = \text{arcs } ?G$ 
  assume  $h: (\forall e \in ?E. \text{head } ?G\ e \in C - \{Inl\ x\} \vee \text{tail } ?G\ e \in C - \{Inl\ x\})$ 
  have vertex-cover (relation-to-digraph  $A\ r$ ) (arcs (relation-to-digraph  $A\ r$ )) ( $C - \{Inl\ x\}$ )
  proof–
    have  $C - \{Inl\ x\} \subseteq \text{verts } (\text{relation-to-digraph } A\ r)$ 
    using assms(1) vertex-cover-def
    by (metis assms(2) insert-Diff insert-subset)
    thus  $?thesis$  using  $h$  vertex-cover-def
    by blast
  qed
  thus  $?thesis$  by auto
qed

lemma matching-vertex-cover:
  assumes minimum-vertex-cover (relation-to-digraph  $A\ r$ ) (arcs (relation-to-digraph  $A\ r$ ))  $C$ 
  and finite  $A$ 
  shows  $Inl\ x \notin C \vee Inr\ x \notin C$ 
proof(rule ccontr)

```

assume $h: \neg (Inl\ x \notin C \vee Inr\ x \notin C)$
hence $vertex-cover\ (relation-to-digraph\ A\ r)\ (arcs\ (relation-to-digraph\ A\ r))$
 $(C - \{Inl\ x\})$
 $\vee\ vertex-cover\ (relation-to-digraph\ A\ r)\ (arcs\ (relation-to-digraph\ A\ r))$
 $(C - \{Inr\ x\})$
using $vertex-cover-subset$ **by** $(metis\ assms(1)\ minimum-vertex-cover-def)$
thus $False$ **using** $assms(2)\ minimum-vertex-cover-def\ h$
by $(metis\ assms(1)\ card-Diff1-less\ diff-shunt-var\ finite.emptyI\ finite-Diff2$
 $finite-verts\ leD\ vertex-cover-def)$
qed

lemma $L-R-disjoint$:

assumes $minimum-vertex-cover\ (relation-to-digraph\ A\ r)\ (arcs\ (relation-to-digraph\ A\ r))\ C$
and $finite\ A$
shows $\{x \in A. Inl\ x \in C\} \cap \{x \in A. Inr\ x \in C\} = \{\}$ **using** $matching-vertex-cover$
 $assms(1,2)$
using $assms(2)$ **by** $fastforce$

lemma $card-projection$:

assumes $minimum-vertex-cover\ (relation-to-digraph\ A\ r)\ (arcs\ (relation-to-digraph\ A\ r))\ C$ **and**
 $finite\ A$
shows

$card\ \{x \in A. Inl\ x \in C \vee Inr\ x \in C\} = card\ C$

proof –

have $Inl\ 'A \cap Inr\ 'A = \{\}$ **by** $auto$

let $?L = \{x \in A. Inl\ x \in C\}$

let $?R = \{x \in A. Inr\ x \in C\}$

have $d: ?L \cap ?R = \{\}$

using $assms(1,2)\ matching-vertex-cover$ **by** $fastforce$

have $S-def$:

$\{x \in A. Inl\ x \in C \vee Inr\ x \in C\} = ?L \cup ?R$ **by** $auto$

have $?L \cap ?R = \{\}$ **using** $L-R-disjoint$ **by** $(simp\ add: d)$

moreover

have $fin1: finite\ ?L$ **and** $fin2: finite\ ?R$

using $assms$ **by** $auto$

ultimately

have $card\ (?L \cup ?R) = card\ ?L + card\ ?R$

by $(meson\ card-Un-disjoint)$

moreover

have $card\ C = card\ ?L + card\ ?R$

proof –

have $C \subseteq Inl\ 'A \cup Inr\ 'A$ **using** $vertex-cover-def$

by $(metis\ (no-types,\ lifting)\ Plus-def\ assms(1)\ minimum-vertex-cover-def\ relation-to-digraph-def\ select-convs(1))$

hence $C = Inl\ ' ?L \cup Inr\ ' ?R$ **by** $auto$

moreover

have $Inl\ ' ?L \cap Inr\ ' ?R = \{\}$ **by** $auto$

```

moreover
have inj-on Inl ?L and inj-on Inr ?R by auto
ultimately show ?thesis using fin1 fin2
  by (metis (no-types, lifting) card-Un-disjoint card-image finite-imageI)
qed
thus ?thesis
  using S-def calculation by argo
qed

lemma cover-antichain:
  assumes minimum-vertex-cover (relation-to-digraph A r) (arcs (relation-to-digraph
A r)) C and
    finite A
  shows anti-chain A r {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C} ∧
    card {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C} = card A − card C
proof(rule conjI)
  show anti-chain A r {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C}
proof−
  have 1: {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C} ⊆ A ∧ partial-order-on A r
    by (simp add: p-o-translation)
  have 2: anti-total {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C} r
proof(rule ccontr)
  assume  $\neg$  anti-total {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C} r
  hence  $(\exists x \in A. \exists y \in A. x \neq y \wedge$ 
     $(\text{Inl } x \in (A <+> A) - C \wedge \text{Inr } x \in (A <+> A) - C) \wedge$ 
     $(\text{Inl } y \in (A <+> A) - C \wedge \text{Inr } y \in (A <+> A) - C) \wedge$ 
     $((x,y) \in r \vee (y,x) \in r))$ 
    using anti-total-def by blast
  then obtain x and y where a:  $((x \in A \wedge y \in A \wedge x \neq y) \wedge$ 
     $(\text{Inl } x \in (A <+> A) - C \wedge \text{Inr } x \in (A <+> A) - C) \wedge$ 
     $(\text{Inl } y \in (A <+> A) - C \wedge \text{Inr } y \in (A <+> A) - C) \wedge$ 
     $((x,y) \in r \vee (y,x) \in r))$ 
    by blast
  hence  $(x \in A \wedge y \in A \wedge (x,y) \in r \wedge x \neq y) \vee (x \in A \wedge y \in A \wedge (y,x) \in r \wedge$ 
x ≠ y )
    by blast
  hence b:  $(\text{Inl } x, \text{Inr } y) \in (\text{arcs } (\text{relation-to-digraph } A \text{ } r)) \vee$ 
     $(\text{Inl } y, \text{Inr } x) \in (\text{arcs } (\text{relation-to-digraph } A \text{ } r))$ 
    using relation-to-digraph-def
    by (smt (verit, best) mem-Collect-eq select-convs(2))
  hence  $(\text{Inl } x = \text{tail } (\text{relation-to-digraph } A \text{ } r) (\text{Inl } x, \text{Inr } y) \wedge$ 
     $\text{Inr } y = \text{head } (\text{relation-to-digraph } A \text{ } r) (\text{Inl } x, \text{Inr } y)) \vee$ 
     $(\text{Inl } y = \text{tail } (\text{relation-to-digraph } A \text{ } r) (\text{Inl } y, \text{Inr } x) \wedge$ 
     $\text{Inr } x = \text{head } (\text{relation-to-digraph } A \text{ } r) (\text{Inl } y, \text{Inr } x))$ 
    by (simp add: relation-to-digraph-def)
  hence  $(\text{Inl } x \in C \vee \text{Inr } y \in C) \vee (\text{Inl } y \in C \vee \text{Inr } x \in C)$ 
    using relation-to-digraph-def[of A r] vertex-cover-def[of relation-to-digraph
A r]
    minimum-vertex-cover-def assms b

```

```

      by (metis (mono-tags, lifting) prod.simps(2) select-convs(3,4))
    thus False using a by auto
  qed
  thus ?thesis
    by (simp add: 1 anti-chain-def)
  qed
next
  let ?X = {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C}
  have partition:
    A = ?X ∪ {x ∈ A. Inl x ∈ C ∨ Inr x ∈ C} by auto
  moreover
  have disj:
    ?X ∩ {x ∈ A. Inl x ∈ C ∨ Inr x ∈ C} = {} by auto
  ultimately
  have card A = card ?X + card {x ∈ A. Inl x ∈ C ∨ Inr x ∈ C}
    using assms(1)
    by (metis (lifting) assms(2) card-Un-disjoint finite-Un)
  moreover
  have card {x ∈ A. Inl x ∈ C ∨ Inr x ∈ C} = card C using card-projection
    assms(1,2) p-o-translation by blast
  ultimately
  show card ?X = card A − card C by simp
qed

lemma antichain-le-chain-decomposition:
  assumes chain-decomposition A r C and anti-chain A r X
  and finite X and finite C
  shows card X ≤ card C
proof-
  have cover:
     $\bigwedge x. x \in X \implies \exists c \in C. x \in c$ 
    using assms(1) anti-chain-def chain-decomposition-def
    by (metis assms(2) UnionE partition-on-def subsetD)
  then obtain f where
    f-def:  $\bigwedge x. x \in X \implies f x \in C \wedge x \in f x$ 
    by metis
  have inj: inj-on f X
  proof (rule inj-onI)
    fix x y
    assume x: x ∈ X and y: y ∈ X and eq: f x = f y
    from f-def[OF x] have x ∈ f x by auto
    moreover
    from f-def[OF y] have y ∈ f y by auto
    ultimately
    have x ∈ f x ∧ y ∈ f x using eq
      by auto
    hence both-in-chain:
      x ∈ f x ∧ y ∈ f x by auto
    have chain (f x) A r

```

```

    using chain-decomposition-def assms(1) f-def x
  by blast
  hence total:total-on (f x) r using chain-def by auto
  have (x,y) ∈ r ∨ (y,x) ∈ r
    using total both-in-chain total-on-def
    by (metis Dilworth-Finite.chain-def ‹Dilworth-Finite.chain (f x) A r› partial-order-onD(1) refl-onD subsetD)
  have x ≠ y ⟶ ¬((x,y) ∈ r ∨ (y,x) ∈ r)
    using assms(2) x y anti-chain-def
    by (metis anti-total-def)
  thus x = y
    using ‹(x,y) ∈ r ∨ (y,x) ∈ r›
    by blast
qed
have f ‘ X ⊆ C
  using f-def by auto
moreover
have finite (f ‘ X)
  using assms(3) by auto
ultimately have card (f ‘ X) ≤ card C using assms(4)
  by (simp add: card-mono)
moreover have card X = card (f ‘ X)
  using inj assms(3)
  by (simp add: card-image)
ultimately show ?thesis by simp
qed
end

```

lemma *Konig-Egervary-relation:*

```

  fixes A:: 'a::countable set
  assumes
    maximum-dirBD-matching (relation-to-digraph A r) (Inl ‘ A) (Inr ‘ A) E and
    minimum-vertex-cover (relation-to-digraph A r) (arcs (relation-to-digraph A
r)) C and
    finite (verts (relation-to-digraph A r))
  shows card E = card C using assms Konig-Egervary
    dirBD-matching-def maximum-dirBD-matching-def by blast

```

lemma (in part-order-countable)exists-max-antichain-from-matching:

```

  assumes finite A and maximum-dirBD-matching (relation-to-digraph A r) (Inl
‘ A) (Inr ‘ A) M
  shows ∃ X. largest-antichain A r X ∧ (card X = card A − card M)
proof −
  have ∃ C. minimum-vertex-cover (relation-to-digraph A r) (arcs (relation-to-digraph
A r)) C
    by (meson assms(1,2) dirBD-matching-def exists-min-vertex-cover finite-Un
finite-imageI
    maximum-dirBD-matching-def)
  then obtain C where

```

```

  Cmin: minimum-vertex-cover (relation-to-digraph A r) (arcs (relation-to-digraph
A r)) C by auto
  have card-eq: card M = card C using Konig-Egervary-relation
  using Cmin assms(1,2) finite-verts by blast
  have fin1: finite C
  by (meson Cmin assms(1) finite-verts minimum-vertex-cover-def rev-finite-subset
vertex-cover-def)
  let ?X = {x ∈ A. Inl x ∉ C ∧ Inr x ∉ C}
  have anti: anti-chain A r ?X ∧ card ?X = card A - card C
  using Cmin assms(1) cover-antichain
  by blast
  have *: ∀ X'. anti-chain A r X' ⟶ card X' ≤ card ?X
  proof(rule allI, rule impI)
    fix X'
    assume h: anti-chain A r X'
    have c: chain-decomposition A r (chain-decomposition-order A r M)
    using chain-decomposition-cardinality assms(1,2) maximum-dirBD-matching-def
p-o-translation
    part-order.exist-chain-decomposition part-order-axioms partial-order-onD(4)
  by blast
  have fin3: finite X' using h
  by (metis anti-chain-def assms(1) finite-subset)
  hence card X' ≤ card (chain-decomposition-order A r M)
  using h antichain-le-chain-decomposition[of chain-decomposition-order A r
M] c
  by (metis assms(1) chain-decomposition-def finite-elements)
  thus card X' ≤ card ?X
  using anti assms(1,2) card-eq maximum-dirBD-matching-def part-order.chain-decomposition-cardinality

    part-order-axioms
  by fastforce
qed
from anti *
have (largest-antichain A r ?X) ∧ (card ?X = card A - card M)
  by (simp add: card-eq largest-antichain-def)
thus ?thesis by auto
qed

lemma (in part-order-countable)smallest-chain-decomposition-order:
  assumes finite A and maximum-dirBD-matching (relation-to-digraph A r) (Inl
' A) (Inr ' A) M
  shows ∃ DC. (smallest-chain-decomposition A r DC) ∧ (card DC = card A -
card M)
proof-
  let ?C = chain-decomposition-order A r M
  have *: chain-decomposition A r ?C ∧ (card ?C = card A - card M)
  using assms chain-decomposition-cardinality maximum-dirBD-matching-def
p-o-translation
    part-order.exist-chain-decomposition part-order-axioms partial-order-onD(4)

```

by *metis*
 moreover
 have $(\forall P. \text{chain-decomposition } A \ r \ P \longrightarrow \text{card } ?C \leq \text{card } P)$
 proof(intro allI impI)
 fix P
 assume $h: \text{chain-decomposition } A \ r \ P$
 have $\exists X. \text{anti-chain } A \ r \ X \wedge \text{card } X = \text{card } A - \text{card } M$
 using *exists-max-antichain-from-matching*
 using *assms(1,2) largest-antichain-def p-o-translation* by *blast*
 then obtain X where $X: \text{anti-chain } A \ r \ X \wedge \text{card } X = \text{card } A - \text{card } M$
 by *auto*
 hence $\text{card } X \leq \text{card } P$
 using *antichain-le-chain-decomposition*
 by (*metis antichain-le-chain-decomposition assms(1) card-eq-0-iff*
chain-decomposition-def finite-elements h less-eq-nat.simps(1))
 thus $\text{card } ?C \leq \text{card } P$ using $* X$ by *argo*
 qed
 ultimately
 show *?thesis* using *smallest-chain-decomposition-def*
 by *blast*
 qed

lemma *chain-unit*:

assumes $r \subseteq A \times A$ and $r \neq \{\}$ and $\forall a \ b. (a, b) \in r \longrightarrow a = b$
 shows $(\text{chain } B \ A \ r) \wedge B \neq \{\} \longrightarrow (\exists x \in A. B = \{x\})$
 proof(rule impI)
 assume *hip*: $\text{chain } B \ A \ r \wedge B \neq \{\}$
 show $\exists x \in A. B = \{x\}$
 proof(rule ccontr)
 assume $\neg (\exists x \in A. B = \{x\})$
 hence $(\forall x \in A. B \neq \{x\})$ by *simp*
 hence $\exists x. \exists y. x \neq y \wedge x \in B \wedge y \in B$
 by (*meson Dilworth-Finite.chain-def hip is-singletonI' is-singleton-some-elem*
some-elem-nonempty subsetD)
 hence $\exists x. \exists y. x \neq y \wedge (x, y) \in r$
 by (*metis Dilworth-Finite.chain-def hip total-on-def*)
 thus *False* using *assms(3)* by *auto*
 qed
 qed

lemma *chain-descomposition-unit*:

assumes $r \subseteq A \times A$ and $\forall a \ b. (a, b) \in r \longrightarrow a = b$ and *partial-order-on* A
 and $r \neq \{\}$
 shows $(\text{chain-decomposition } A \ r \ \{\{x\} \mid x. x \in A\}) \wedge ((\text{chain-decomposition } A \ r \ D) \longrightarrow D = \{\{x\} \mid x. x \in A\})$
 proof(rule conjI)
 show *chain-decomposition* $A \ r \ \{\{x\} \mid x. x \in A\}$
 proof(*unfold chain-decomposition-def, rule conjI*)
 show *partition-on* $A \ \{\{x\} \mid x. x \in A\}$

```

    by (simp add: partition-on-singletons setcompr-eq-image)
  next
    show  $\forall B \in \{\{x\} \mid x. x \in A\}. \text{chain } B \ A \ r$ 
    using Dilworth-Finite.chain-def assms(3) by fastforce
  qed
next
show chain-decomposition  $A \ r \ D \longrightarrow D = \{\{x\} \mid x. x \in A\}$ 
proof(rule impI)
  assume *: chain-decomposition  $A \ r \ D$ 
  show  $D = \{\{x\} \mid x. x \in A\}$ 
  proof(rule ccontr)
    let ?D1 =  $\{\{x\} \mid x. x \in A\}$ 
    assume hip:  $D \neq ?D1$ 
    show False
    proof(rule disjE)
      show  $\neg(D \subseteq ?D1) \vee \neg(?D1 \subseteq D)$  using hip by simp
    next
      assume  $\neg D \subseteq ?D1$ 
      hence  $\exists B. B \in D \wedge B \notin ?D1$  by auto
      then obtain B where B:  $B \in D \wedge B \notin ?D1$  by auto
      hence  $\exists x. \exists y. x \neq y \wedge x \in B \wedge y \in B$  using *
      by (smt (verit, del-insts) chain-decomposition-def is-singletonI' is-singleton-def
        mem-Collect-eq mem-simps(9) partition-on-def singletonI)
      hence  $\exists x. \exists y. x \neq y \wedge (x, y) \in r$  using *
      by (metis (no-types, opaque-lifting) B Dilworth-Finite.chain-def chain-decomposition-def
        total-on-def)
      thus False using assms(2) by auto
    next
      assume  $\neg ?D1 \subseteq D$ 
      hence  $\exists x. \{x\} \in ?D1 \wedge \{x\} \notin D$  by auto
      thus False using * assms chain-unit
      by (smt (verit) UnionE chain-decomposition-def mem-Collect-eq partial-order-onD(4) partition-on-def
        singletonD)
    qed
  qed
qed
qed
qed

lemma antichain-on:
  assumes  $r \subseteq A \times A$  and  $\forall a \ b. (a, b) \in r \longrightarrow a = b$  and partial-order-on  $A$ 
  and
     $r \neq \{\}$ 
  shows anti-chain  $A \ r \ A$ 
  using anti-total-def anti-chain-def assms by (metis subsetI)

lemma largest-antichain-on:
  assumes  $r \subseteq A \times A$  and  $\forall a \ b. (a, b) \in r \longrightarrow a = b$  and partial-order-on  $A$ 

```

r and
 $r \neq \{\}$ and finite A
shows largest-antichain A r A **using** anti-total-def anti-chain-def largest-antichain-def

proof–
 have anti-chain A r A **using** assms antichain-on **by** blast
 moreover
 have $\forall B. \text{anti-chain } A \ r \ B \longrightarrow \text{card } B \leq \text{card } A$
proof(rule allI, rule impI)
 fix B
 assume anti-chain A r B
 hence $B \subseteq A$ **using** anti-chain-def[of A r B] **by** simp
 thus $\text{card } B \leq \text{card } A$ **using** assms(5)
 by (simp add: card-mono)
 qed
 ultimately
 show ?thesis **using** largest-antichain-def **by** auto
 qed

lemma card-smCD-lgAC1:
 assumes finite A **shows** $\text{card } A = \text{card } \{\{x\} \mid x. x \in A\}$
using assms **by** (metis Setcompr-eq-image inj-on-iff-eq-card inj-singleton)

lemma arcs-empty:
 assumes $r \subseteq A \times A$ **and** arcs (relation-to-digraph A r) = $\{\}$
 shows $\forall a \ b. (a, b) \in r \longrightarrow a = b$
proof(rule ccontr)
 assume $\neg (\forall a \ b. (a, b) \in r \longrightarrow a = b)$
 thus False
 by (smt (z3) Collect-empty-eq assms(1,2) mem-Sigma-iff relation-to-digraph-def
 select-convs(2) subset-eq)
 qed

lemma card-smCD-lgAC:
 assumes $r \subseteq A \times A$ **and** $\forall a \ b. (a, b) \in r \longrightarrow a = b$ **and** partial-order-on A
 r and
 $r \neq \{\}$ **and** finite A **and**
 smallest-chain-decomposition A r smCD **and** largest-antichain A r lgAC
shows $\text{card smCD} = \text{card lgAC}$
using assms largest-antichain-on card-smCD-lgAC1
by (metis (no-types, lifting) anti-chain-def card-seteq chain-descomposition-unit
 largest-antichain-def
 partial-order-onD(4) smallest-chain-decomposition-def)

lemma (in part-order)Dilworth-empty:
 assumes arcs (relation-to-digraph A r) = $\{\}$ **and** finite A **and**
 smallest-chain-decomposition A r smCD **and** largest-antichain A r lgAC
shows $\text{card smCD} = \text{card lgAC}$

```

proof–
  have *:  $\forall a\ b. (a, b) \in r \longrightarrow a = b$  using arcs-empty
  by (metis assms(1) part-order.p-o-translation part-order-axioms partial-order-onD(4))
  show ?thesis
  proof(cases  $r = \{\}$ )
    assume  $r = \{\}$ 
    hence  $A = \{\}$ 
    using p-o-translation partial-order-onD(1) refl-onD by fastforce
    hence  $smCD = \{\} \wedge lgAC = \{\}$ 
    by (metis anti-chain-def assms(3,4) bot.extremum-uniqueI chain-decomposition-def

      largest-antichain-def partition-on-empty
      smallest-chain-decomposition-def)
    thus ?thesis
    by simp
  next
    assume  $r \neq \{\}$ 
    thus ?thesis
    using assms * card-smCD-lgAC anti-chain-def[of  $A\ r$ ] largest-antichain-def[of
   $A\ r\ lgAC$ ]
    by (metis partial-order-onD(4))
  qed
qed

theorem (in part-order-countable) Dilworth-Finite:
  assumes finite  $A$ 
  shows  $\exists smCD. \exists lgAC. \text{smallest-chain-decomposition } A\ r\ smCD \wedge \text{largest-antichain}$ 
 $A\ r\ lgAC$ 
 $\wedge \text{card } smCD = \text{card } lgAC$ 
proof(rule disjE)
  show  $\text{arcs } (\text{relation-to-digraph } A\ r) = \{\} \vee \text{arcs } (\text{relation-to-digraph } A\ r) \neq \{\}$ 
by auto
next
  assume  $\text{arcs } (\text{relation-to-digraph } A\ r) = \{\}$ 
  thus ?thesis using Dilworth-empty
  using assms
  by (metis assms  $\langle \text{arcs } (\text{relation-to-digraph } A\ r) = \{\} \rangle$  Dilworth-empty
    smallest-chain-decomposition-order exists-max-antichain-from-matching part-order-axioms

    partial-order-onD(4) finite-arcs rel-to-digraph-is-dir-bipartite maximum-matching-digraph

    part-order.p-o-translation)
next
  assume hip:  $\text{arcs } (\text{relation-to-digraph } A\ r) \neq \{\}$ 
  have dir-bipartite-digraph  $(\text{relation-to-digraph } A\ r)$  (Inl ‘  $A$ ) (Inr ‘  $A$ )
    using assms rel-to-digraph-is-dir-bipartite p-o-translation
    by (metis partial-order-onD(4))
  moreover
  have finite  $(\text{arcs } (\text{relation-to-digraph } A\ r))$  using finite-arcs assms

```

```

    by (metis p-o-translation partial-order-onD(4))
  ultimately
  have  $\exists M. \text{maximum-dirBD-matching } (\text{relation-to-digraph } A \ r) \ (\text{Inl } 'A) \ (\text{Inr } 'A) \ M$ 
  using maximum-matching-digraph hip by blast
  then obtain  $M$  where  $M: \text{maximum-dirBD-matching } (\text{relation-to-digraph } A \ r) \ (\text{Inl } 'A) \ (\text{Inr } 'A) \ M$ 
  by auto
  hence  $DC: (\exists DC. (\text{smallest-chain-decomposition } A \ r \ DC) \wedge (\text{card } DC = \text{card } A - \text{card } M))$ 
  using assms(1) smallest-chain-decomposition-order by blast
  moreover
  have  $(\exists AC. (\text{largest-antichain } A \ r \ AC) \wedge (\text{card } AC = \text{card } A - \text{card } M))$ 
  using assms(1) M exists-max-antichain-from-matching
  by blast
  ultimately
  show ?thesis by metis
qed
end

```

4 Dilworth Countable Theorem

```

theory DilworthCountable
  imports
    Dilworth-Finite
    Prop-Compactness.k-coloring
begin

```

The countable infinite version of Dilworth's theorem for partially ordered sets states that, whenever the "width" of the partial order is finite, i.e., whenever there exists a finite upper bound on the size of anti-chains, there exists a finite minimal chain decomposition with size equal to the size of the largest anti-chain.

definition *incomparability-graph* :: $'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ digraph}$

where

$\text{incomparability-graph } r \ A \equiv (A, \{(x, y) \mid x \ y. \ x \in A \wedge y \in A \wedge (x, y) \notin r \wedge (y, x) \notin r \wedge x \neq y\})$

lemma *chain-decomposition-colorable*:

assumes *partition-on* $A \ P$ **and** *finite* P **and** $(\forall B \in P. (\text{total-on } B \ r))$

shows $(\text{colorable } (\text{incomparability-graph } r \ A) \ (\text{card } P))$

proof–

let $?k = (\text{card } P)$

have $\exists f. P = f \ ' \{i. i < ?k\} \wedge \text{inj-on } f \ \{i. i < ?k\}$ **using** *assms(2)*

by (*metis card-Collect-less-nat card-image finite-imp-nat-seg-image-inj-on*)

then obtain f **where** $f1: P = f \ ' \{i::\text{nat}. i < ?k\}$ **and** $f2: \text{inj-on } f \ \{i. i < ?k\}$

by *auto*

```

let ?G = incomparability-graph r A
let ?c = λv. THE i. i < ?k ∧ (∃ p ∈ P. v ∈ p ∧ f i = p)
have *: ∀ u. u ∈ V[?G] → (∃ !i. i < ?k ∧ (∃ p. p ∈ P ∧ u ∈ p ∧ f i = p) ) ∧ ?c(u)
= i)
proof(intro allI impI)
  fix u
  assume hip: u ∈ V[?G]
  show (∃ !i. i < ?k ∧ (∃ p. p ∈ P ∧ u ∈ p ∧ f i = p) ) ∧ ?c(u) = i)
  proof–
    have 1: ∃ i. i < ?k ∧ (∃ p. p ∈ P ∧ u ∈ p ∧ f i = p) using hip assms f1
    by(metis (no-types, lifting) UN-E image-eqI
      incomparability-graph-def mem-Collect-eq partition-onD1 prod.sel(1))
    then obtain p i where p: p ∈ P ∧ u ∈ p and i1: f i = p and i2: i < ?k
by blast
    hence i < ?k ∧ (∃ p. p ∈ P ∧ u ∈ p ∧ f i = p) by auto
    have 2: ∀ x. x < ?k ∧ (∃ p ∈ P. u ∈ p ∧ f x = p) → x = i
    proof(intro allI impI)
      fix x
      assume hip: x < ?k ∧ (∃ p ∈ P. u ∈ p ∧ f x = p)
      show x = i
      proof–
        from hip obtain q where q: q ∈ P ∧ u ∈ q and q1: f x = q by auto
        hence 1: f x ∈ f ‘ {i::nat. i < ?k} using f1 by auto
        have 2: f i ∈ f ‘ {i::nat. i < ?k} using i2 by auto
        have p = q using q p assms(1) using disjointD partition-on-def by
fastforce
        hence f x = f i using q1 i1 by auto
        hence x = i using hip i2 1 2 f2 by(simp add: inj-onD)
        thus ?thesis by auto
      qed
    qed
    from 1 and 2 show ?thesis by auto
    have 3: ?c(u) = i using 1 2 p by auto
    qed
  qed
have c1: (∀ u. u ∈ V[?G] → ?c(u) < ?k) using * by blast
have c2: (∀ u v. (u,v) ∈ E[?G] → ?c(u) ≠ ?c(v))
proof(intro allI impI)
  fix u v
  assume hip: (u,v) ∈ E[?G]
  hence a: u ∈ V[?G] and b: v ∈ V[?G] by(unfold incomparability-graph-def, auto)

  have ∃ i. ∃ p. p ∈ P ∧ u ∈ p ∧ f i = p ∧ i < ?k using * a by auto
  then obtain p i where p: p ∈ P ∧ u ∈ p and i1: f i = p and i2: i < ?k
  and i3: ?c(u) = i using * a by auto
  have ∃ j. ∃ q. q ∈ P ∧ v ∈ q ∧ f j = q ∧ j < ?k using * b by auto
  then obtain q j where q: q ∈ P ∧ v ∈ q and j1: f j = q and j2: j < ?k
  and j3: ?c(v) = j using * b by auto
  show ?c(u) ≠ ?c(v)

```

```

proof(rule notI)
  assume ?c(u) = ?c(v)
  hence  $\exists$ :  $p = q$  using i1 i3 j1 j3 by auto
  have  $4$ :  $u \in A$  and  $v \in A$  using hip by(unfold incomparability-graph-def, auto)
  hence  $(u,v) \in r \vee (v,u) \in r$  using p q  $\exists$  assms( $\exists$ )
  by (smt (verit, best) CollectD Pair-inject hip incomparability-graph-def sndI
total-on-def)
  thus False using hip by(unfold incomparability-graph-def, auto)
qed
qed
hence coloring ?c ?k (incomparability-graph r A)
  using c1 c2 by(unfold coloring-def, auto)
thus ?thesis using c1 c2 by (unfold colorable-def, auto)
qed

```

```

lemma non-empty-set-color: assumes coloring c k (incomparability-graph r A)
and  $j \in c'A$ 
shows  $\{x \in A. c\ x = j\} \neq \{\}$ 
proof-
  have  $V[\text{incomparability-graph } r\ A] = A$ 
  using incomparability-graph-def[of r A] by auto
  have  $\exists x \in A. c(x) = j$ 
  using assms incomparability-graph-def[of r A] by auto
  thus ?thesis by auto
qed

```

```

lemma coloring-particion:
  assumes coloring c k (incomparability-graph r A)
  shows partition-on A  $\{\{x \in A. c(x) = j\} \mid j. j \in c'A\}$ 
proof-
  let ?G = incomparability-graph r A
  have 1:  $\bigcup \{\{x \in A. c\ x = j\} \mid j. j \in c'(V[?G])\} = A$ 
  proof-
    have  $\bigcup \{\{x \in A. c\ x = j\} \mid j. j \in c'(V[?G])\} \subseteq A$  by auto
    moreover
    have  $A \subseteq \bigcup \{\{x \in A. c\ x = j\} \mid j. j \in c'(V[?G])\}$ 
  proof
    fix x
    assume h:  $x \in A$ 
    hence  $x \in V[?G]$  using incomparability-graph-def
    by (metis (lifting) fst-conv)
    hence  $\exists j. c(x) = j$  using assms coloring-def
    by metis
    then obtain j where  $j: c(x) = j$  by auto
    hence  $x \in \{x \in A. c\ x = j \mid j \in c'(V[?G])\}$  using h by auto
    thus  $x \in \bigcup \{\{x \in A. c\ x = j\} \mid j. j \in c'(V[?G])\}$  using j
    using  $\langle x \in V[\text{incomparability-graph } r\ A] \rangle$  by blast
  qed
finally

```

```

    show  $\bigcup \{ \{x \in A. c\ x = j\} \mid j. j \in c'(V[?G]) \} = A$  by auto
  qed
  have 2: disjoint  $\{ \{x \in A. c\ x = j\} \mid j. j \in c'(V[?G]) \}$ 
    using assms coloring-def
    by (smt (verit, best) disjnt-iff mem-Collect-eq pairwiseI)
  have 3:  $\{\} \notin \{ \{x \in A. c\ x = j\} \mid j. j \in c'(V[?G]) \}$ 
    using assms coloring-def non-empty-set-color
    by (smt (verit, best) Collect-empty-eq fst-conv incomparability-graph-def mem-Collect-eq)

  show ?thesis using 1 2 3
    by (simp add: incomparability-graph-def partition-on-def)
  qed

lemma (in part-order) coloring-total:
  assumes coloring c k (incomparability-graph r A) and  $j \in c'A$ 
  shows total-on  $\{x \in A. c\ x = j\}$  r
proof -
  {
    fix x y
    assume h1:  $x \in \{x \in A. c\ x = j\}$  and h2:  $y \in \{x \in A. c\ x = j\}$ 
    have  $x \neq y \longrightarrow (x, y) \in r \vee (y, x) \in r$ 
    proof
      assume  $x \neq y$ 
      thus  $(x, y) \in r \vee (y, x) \in r$  using h1 h2 assms incomparability-graph-def [of
r A]
        by (smt (verit, best) coloring-def mem-Collect-eq snd-conv)
    qed
  }
  thus ?thesis using total-on-def
    by blast
qed

lemma partition-card-le-colors:
  assumes col: coloring c k G
  shows card  $(\{ \{x \in A. c\ x = j\} \mid j. j \in c'(V[G]) \}) \leq k$ 
proof (cases  $k = 0$ )
  assume  $k = 0$ 
  hence  $V[G] = \{\}$  using assms coloring-def
    by (meson coloring-nonemptygraph colorable-def dual-order.irrefl)
  thus ?thesis
    by simp
next
  assume  $k \neq 0$ 
  hence  $k > 0$  by auto
  let ?P =  $\{ \{x \in A. c\ x = j\} \mid j. j \in c'(V[G]) \}$ 
  have P-image: ?P =  $(\lambda j. \{x \in A. c\ x = j\})'(c'(V[G]))$ 
    by auto
  have fin-colors: finite  $(c'(V[G]))$ 
  proof -

```

```

have card (c'(V[G])) ≤ k
  using coloring-card-image[OF col] by simp
thus ?thesis
  by (metis (no-types, lifting) bounded-nat-set-is-finite col coloring-def imageE)
qed
have card ?P ≤ card (c' (V[G]))
  unfolding P-image using fin-colors
  by (rule card-image-le)
also have ... ≤ k
  using coloring-card-image[OF col] by simp
finally show ?thesis by simp
qed

lemma (in part-order) coloring-chain-decomposition1:
  assumes coloring c k (incomparability-graph r A)
  shows chain-decomposition A r { {x ∈ A. c x = j} | j. j ∈ c'A } ∧
    card ({ {x ∈ A. c x = j} | j. j ∈ c'A }) ≤ k
proof -
  have chain-decomposition A r { {x ∈ A. c x = j} | j. j ∈ c'A }
    using coloring-particion[of c k r A] coloring-total[of c k]
    by (smt (verit, best) Dilworth-Finite.chain-def Sup-upper assms(1)
      chain-decomposition-def mem-Collect-eq p-o-translation partition-onD1)
  moreover
  have card ({ {x ∈ A. c x = j} | j. j ∈ c'A }) ≤ k using assms partition-card-le-colors
    coloring-def[of c k (incomparability-graph r A)]
    coloring-card-image[of c k (incomparability-graph r A)]
    using incomparability-graph-def[of r A] by fastforce
  ultimately show ?thesis by auto
qed

lemma coloring-chain-decomposition-finite:
  assumes coloring c k (incomparability-graph r A)
  shows finite ({ {x ∈ A. c x = j} | j. j ∈ c'A })
proof -
  let ?CD = { {x ∈ A. c x = j} | j. j ∈ c'A }
  have ?CD = (λj. {x ∈ A. c x = j}) ' (c'A)
    by auto
  moreover have finite (c'A)
  proof -
    have c'A ⊆ {0..

```

qed

lemma (in *part-order*) *coloring-chain-decomposition*:

assumes *coloring* *c* *k* (*incomparability-graph* *r* *A*)

shows $(\exists CD. \text{chain-decomposition } A \text{ } r \text{ } CD \wedge \text{card } CD \leq k \wedge \text{finite } CD)$

using *coloring-chain-decomposition1* *coloring-chain-decomposition-finite* *assms*

by *blast*

lemma (in *part-order-countable*) *width-coloring-finite*:

assumes $(\forall AC. \text{anti-chain } A \text{ } r \text{ } AC \longrightarrow \text{card } AC \leq m)$ **and** *finite* *A*

shows $(\exists c. \exists k. \text{coloring } c \text{ } k \text{ } (\text{incomparability-graph } r \text{ } A) \wedge k \leq m)$

proof –

have $\exists sCD. \text{chain-decomposition } A \text{ } r \text{ } sCD \wedge \text{card } sCD \leq m$

by (*metis* *assms*(2) *assms*(1) *smallest-chain-decomposition-def* *largest-antichain-def*

Dilworth-Finite)

then obtain *sCD* **where** *sCD*: *chain-decomposition* *A* *r* *sCD* $\wedge \text{card } sCD \leq m$

by *auto*

hence $1:(\text{colorable } (\text{incomparability-graph } r \text{ } A) \text{ } (\text{card } sCD))$

using *chain-decomposition-def* *chain-decomposition-colorable*

by (*metis* *Dilworth-Finite.chain-def* *assms*(2) *finite-elements*)

have $\exists c. \text{coloring } c \text{ } (\text{card } sCD) \text{ } (\text{incomparability-graph } r \text{ } A) \wedge \text{card } sCD \leq m$

using *colorable-def* 1 *sCD*

by *blast*

thus *?thesis*

using *colorable-def* *coloring-def* *sCD*

by (*smt* (*verit*, *best*) $\langle \text{chain-decomposition } A \text{ } r \text{ } sCD \wedge \text{card } sCD \leq m \rangle$ *coloring-def* *order-trans*)

qed

lemma (in *part-order-countable*) *partial-order-restrc*:

assumes $B \subseteq A$

shows *partial-order-on* *B* (*Restr* *r* *B*)

using *partial-order-on-def*[*of* *B* (*Restr* *r* *B*)]

by (*smt* (*verit*, *best*) *Int-iff* *Sigma-cong* *antisym-Restr* *assms* *mem-Sigma-iff* *p-o-translation*

partial-order-on-def *preorder-on-def* *refl-on-def* *subsetD* *subsetI* *trans-Restr*)

lemma (in *part-order-countable*) *antichain-restrc*:

assumes *anti-chain* *A* *r* *AC* **and** $B \subseteq A$

shows *anti-chain* *B* (*Restr* *r* *B*) $(AC \cap B)$

using *partial-order-restrc*[*of* *B*] *anti-chain-def*[*of* *B* (*Restr* *r* *B*) $(AC \cap B)$]

by (*meson* *IntE* *anti-chain-def* *anti-total-def* *assms*(1,2) *inf-le2*)

lemma (in *part-order-countable*) *width-induced-subgraph*:

assumes $\forall AC. \text{anti-chain } A \text{ } r \text{ } AC \longrightarrow \text{card } AC \leq w$ **and** *finite* *B*

shows $\forall ACB. \text{anti-chain } B \text{ } (\text{Restr } r \text{ } B) \text{ } ACB \longrightarrow \text{card } ACB \leq w$ **using** *assms*

by (*smt* (*verit*, *del-insts*) *Int-iff* *anti-chain-def* *anti-total-def* *mem-Sigma-iff*

p-o-translation *partial-order-onD*(1,4) *refl-on-def* *subset-iff*)

lemma *graph-incomparability*:
is-graph (*incomparability-graph* r A)
using *is-graph-def* *incomparability-graph-def*
by (*smt* (*verit*, *ccfv-SIG*) *mem-Collect-eq* *split-pairs2*)

lemma *induced-subgraph-incomparability*:
assumes *is-induced-subgraph* H (*incomparability-graph* r A) $\wedge$ *finite-graph* H
shows $\exists B. (\text{finite } B) \wedge B = V[H] \wedge (E[H] = E[(\text{incomparability-graph } r A)] \cap (B \times B)) \wedge B \subseteq A$
using *assms is-induced-subgraph-def incomparability-graph-def*
by (*metis* (*mono-tags*, *lifting*) *finite-graph-def fst-conv*)

lemma *incomparability-subgraph*:
assumes *is-graph* (*incomparability-graph* r A)
and *is-subgraph* H (*incomparability-graph* r A) $\wedge$ *finite-graph* H
shows $\exists B. (\text{finite } B) \wedge B = V[H] \wedge (E[H] \subseteq E[(\text{incomparability-graph } r A)] \cap (B \times B)) \wedge B \subseteq A$
using *assms is-subgraph-def incomparability-graph-def*
by (*metis* (*no-types*, *lifting*) *Sigma-cong* *finite-graph-def fst-conv*)

lemma (*in part-order-countable*) *colorable-induced-subgraph*:
assumes $\forall AC. \text{anti-chain } A \ r \ AC \longrightarrow \text{card } AC \leq m$
shows $(\forall H. \text{is-induced-subgraph } H \ (\text{incomparability-graph } r A) \wedge \text{finite-graph } H \longrightarrow \text{colorable } H \ m)$
proof–
have $\forall H. \text{is-induced-subgraph } H \ (\text{incomparability-graph } r A) \wedge \text{finite-graph } H \longrightarrow \text{colorable } H \ m$
proof (*rule allI*, *rule impI*)
fix H
assume $H\text{-assms}: \text{is-induced-subgraph } H \ (\text{incomparability-graph } r A) \wedge \text{finite-graph } H$
have $\exists B. (\text{finite } B) \wedge B = V[H] \wedge (E[H] = E[(\text{incomparability-graph } r A)] \cap (B \times B)) \wedge B \subseteq A$
using *incomparability-subgraph assms(1)*
by (*simp add: H-assms graph-incomparability induced-subgraph-incomparability*)
then obtain B **where** $B: (\text{finite } B) \wedge B = V[H] \wedge (E[H] = E[(\text{incomparability-graph } r A)] \cap (B \times B)) \wedge B \subseteq A$
by *auto*
have $AC\text{-}B: \forall AC. \text{anti-chain } B \ (\text{Restr } r B) \ AC \longrightarrow \text{card } AC \leq m$
using B *antichain-restrc*
by (*smt* (*verit*, *ccfv-threshold*) *Int-iff* *anti-chain-def* *anti-total-def* *assms mem-Sigma-iff* *p-o-translation subset-iff*)
have $poB: \text{partial-order-on } B \ (\text{Restr } r B)$
using *partial-order-restrc* B *p-o-translation*
by *auto*
have $\exists c. \exists k. \text{coloring } c \ k \ (\text{incomparability-graph } (\text{Restr } r B) B) \wedge k \leq m$
using $AC\text{-}B$ B *part-order-countable.width-coloring-finite* *part-order-countable-def*

```

part-order-def poB by blast
  then obtain c k where c: coloring c k (incomparability-graph (Restr r B) B)
 $\wedge k \leq m$  by auto
  hence coloring c k H
    using B induced-subgraph-incomparability
    by (simp add: coloring-def incomparability-graph-def)
  hence coloring c m H using coloring-def
    by (metis c inf.absorb-iff2 inf.strict-boundedE)
  thus colorable H m
    using colorable-def by blast
qed
thus ?thesis
  by blast
qed

lemma exists-smallest-chain-decomposition:
  assumes  $\exists C. \text{chain-decomposition } A \text{ } r \text{ } C$ 
  shows  $\exists C. \text{smallest-chain-decomposition } A \text{ } r \text{ } C$ 
proof-
  let ?Q =  $\lambda n. \exists C. \text{chain-decomposition } A \text{ } r \text{ } C \wedge \text{card } C = n$ 
  obtain k where k1: ?Q k and k2:  $\forall m < k. \neg ?Q m$ 
    using assms ex-least-nat-le[of ?Q] by blast
  obtain C where CD1: chain-decomposition A r C and CD2: card C = k
    using k1 by blast
  have minC:  $\forall P. \text{chain-decomposition } A \text{ } r \text{ } P \longrightarrow \text{card } C \leq \text{card } P$ 
  proof(rule allI)
    fix P
    show chain-decomposition A r P  $\longrightarrow \text{card } C \leq \text{card } P$ 
    proof
      assume PCD: chain-decomposition A r P
      show card C  $\leq \text{card } P$ 
      proof-
        have 1:  $\neg \text{card } P < k$ 
        proof
          assume card P < k
          hence ?Q (card P)
            using PCD by blast
          with k2 show False
          by (metis k2  $\langle \exists C. \text{chain-decomposition } A \text{ } r \text{ } C \wedge \text{card } C = \text{card } P \rangle \langle \text{card } P < k \rangle$ )
        qed
        thus card C  $\leq \text{card } P$  using CD2 1 not-le-imp-less by blast
      qed
    qed
  qed
  have smallest-chain-decomposition A r C
    unfolding smallest-chain-decomposition-def
    using CD1 minC by blast
  thus ?thesis by blast

```

qed

lemma (in *part-order-countable*) *exists-largest-antichain*:

assumes $\forall AC. \text{anti-chain } A \text{ } r \text{ } AC \longrightarrow \text{card } AC \leq m$

shows $\exists lgAC. \text{largest-antichain } A \text{ } r \text{ } lgAC$

proof –

let $?P = \{\text{card } B \mid B. \text{anti-chain } A \text{ } r \text{ } B\}$

have $1: ?P \subseteq \{0..m\}$

using *assms(1) atLeastAtMost-iff* **by** *blast*

moreover

have $\text{anti-chain } A \text{ } r \text{ } \{\}$

using *anti-chain-def anti-total-def p-o-translation*

by *blast*

hence $2: ?P \neq \{\}$

by *blast*

ultimately

have $\exists k. k = \text{Max } ?P$

by *blast*

then obtain k **where** $k = \text{Max } ?P$ **by** *auto*

moreover

have $\text{finP}: \text{finite } ?P$

using 1 **by** (*meson finite-atLeastAtMost finite-subset*)

hence $k\text{-in}: k \in ?P$

using *Max-in calculation 2* **by** *auto*

then obtain $lgAC$ **where**

$lgAC\text{-antichain}: \text{anti-chain } A \text{ } r \text{ } lgAC$ **and** $lgAC\text{-card}: \text{card } lgAC = k$

by *blast*

have $\text{largest}: \text{largest-antichain } A \text{ } r \text{ } lgAC$

proof(*unfold largest-antichain-def, intro conjI allI impI*)

show $\text{anti-chain } A \text{ } r \text{ } lgAC$

by (*rule lgAC-antichain*)

next

fix C

assume $C\text{anti}: \text{anti-chain } A \text{ } r \text{ } C$

have $\text{card } C \in ?P$

using $C\text{anti}$ **by** *blast*

hence $\text{card } C \leq k$

using finP calculation **by** *auto*

hence $\text{card } C \leq \text{card } lgAC$

using $lgAC\text{-card}$ **by** *simp*

thus $\text{card } C \leq \text{card } lgAC$

by *metis*

qed

thus $\exists lgAC. \text{largest-antichain } A \text{ } r \text{ } lgAC$ **by** *auto*

qed

theorem (in *part-order-countable*) *Dilworth-countable-aux*:

assumes $\text{largest-antichain } A \text{ } r \text{ } lgAC$

shows $(\exists CD. (\text{chain-decomposition } A \text{ } r \text{ } CD \wedge \text{finite } CD) \wedge \text{card } CD \leq \text{card } lgAC)$

```

lgAC)
proof -
  have 1: is-graph (incomparability-graph r A)
    using graph-incomparability by auto
  have (colorable (incomparability-graph r A) (card lgAC))
  proof -
    have  $\forall H. \text{is-induced-subgraph } H \text{ (incomparability-graph r A) } \wedge \text{finite-graph } H$ 
     $\longrightarrow$ 
      colorable H (card lgAC)
    using colorable-induced-subgraph assms
    by (metis largest-antichain-def)
  thus ?thesis
    using 1 deBruijn-Erdos-coloring-for-finite-induced-subgraphs
    by metis
qed
thus ?thesis
  using colorable-def coloring-chain-decomposition
  by metis
qed

definition largest-finite-antichain :: 'a set  $\Rightarrow$  'a rel  $\Rightarrow$  'a set  $\Rightarrow$  bool
where largest-finite-antichain A r B  $\equiv$ 
  anti-chain A r B  $\wedge$  finite B  $\wedge$  ( $\forall C. \text{anti-chain } A \text{ r } C \wedge \text{finite } C \longrightarrow \text{card } C$ 
 $\leq \text{card } B$ )

lemma (in part-order-countable) exists-largest-finite-antichain:
  assumes H:  $\forall AC. \text{anti-chain } A \text{ r } AC \wedge \text{finite } AC \longrightarrow \text{card } AC \leq m$ 
  shows  $\exists \text{lgAC}. \text{largest-finite-antichain } A \text{ r } \text{lgAC}$ 
proof -
  let ?P = {card B | B. anti-chain A r B  $\wedge$  finite B}
  have subsetP: ?P  $\subseteq$  {0..m}
  proof -
    fix n
    assume n  $\in$  ?P
    then obtain B where
      B: anti-chain A r B  $\wedge$  finite B and n: n = card B
    by blast
    have card B  $\leq$  m
    using H B by blast
    thus n  $\in$  {0..m}
    using n by auto
  qed
  have anti-empty: anti-chain A r {}
  unfolding anti-chain-def anti-total-def
  using p-o-translation by auto
  hence 2: ?P  $\neq$  {}
  by auto
  have finP: finite ?P
  using subsetP finite-atLeastAtMost

```

```

    by (rule finite-subset)
  let ?k = Max ?P
  have k-in: ?k ∈ ?P
    using 2 Max-eq-iff finP by blast
  then obtain lgAC where
    lgAC1: anti-chain A r lgAC ∧ finite lgAC and lgAC2: card lgAC = ?k
    by fastforce
  have largest:
    largest-finite-antichain A r lgAC
  proof (unfold largest-finite-antichain-def, intro conjI allI impI)
    show anti-chain A r lgAC
      using lgAC1 by blast
    show finite lgAC
      using lgAC1 by blast
  fix C
  assume C: anti-chain A r C ∧ finite C
  have card C ∈ ?P
    using C by blast
  hence card C ≤ ?k
    using finP k-in by simp
  thus card C ≤ card lgAC
    using lgAC2 by simp
qed
show ?thesis
  using largest by blast
qed

```

definition *finite-smallest-chain-decomposition* :: 'a set ⇒ 'a rel ⇒ 'a set set ⇒ bool
where
finite-smallest-chain-decomposition A r CD ≡
chain-decomposition A r CD ∧
finite CD ∧
(∀ P. *chain-decomposition* A r P ∧ *finite* P ⟶ card CD ≤ card P)

lemma *exists-smallest-finite-chain-decomposition*:
assumes ∃ CD. *chain-decomposition* A r CD ∧ *finite* CD
shows ∃ sCD. *finite-smallest-chain-decomposition* A r sCD
proof –
 let ?Q = λn. ∃ C. *chain-decomposition* A r C ∧ *finite* C ∧ card C = n
obtain k **where** k1: ?Q k **and** k2: ∀ m < k. ¬ ?Q m
 using assms ex-least-nat-le[of ?Q] by blast
obtain sC **where** CD1: *chain-decomposition* A r sC ∧ *finite* sC **and** CD2: card
sC = k
 using k1 by blast
have minC: ∀ P. *chain-decomposition* A r P ∧ *finite* P ⟶ card sC ≤ card P
proof(intro allI impI)
 fix P

```

assume PCD: chain-decomposition  $A \ r \ P \wedge \text{finite } P$ 
show  $\text{card } sC \leq \text{card } P$ 
proof–
  have  $1: \neg \text{card } P < k$ 
  proof
    assume  $\text{card } P < k$ 
    hence  $?Q (\text{card } P)$ 
    using PCD by blast
    with  $k2$  show False
    by (metis  $\langle \exists C. \text{chain-decomposition } A \ r \ C \wedge \text{finite } C \wedge \text{card } C = \text{card } P \rangle k2 \langle \text{card } P < k \rangle$ )
  qed
  thus  $\text{card } sC \leq \text{card } P$  using CD2 1 not-le-imp-less by blast
qed
qed
have finite-smallest-chain-decomposition  $A \ r \ sC$ 
unfolding finite-smallest-chain-decomposition-def
using CD1 minC by blast
thus ?thesis by blast
qed

theorem (in part-order-countable) Dilworth-countable:
  assumes  $\exists m. \forall AC. \text{anti-chain } A \ r \ AC \wedge \text{finite } AC \longrightarrow \text{card } AC \leq m$ 
  shows  $\exists smD. \exists lgAC. \text{finite-smallest-chain-decomposition } A \ r \ smD \wedge \text{largest-finite-antichain } A \ r \ lgAC \wedge$ 
     $\text{card } smD = \text{card } lgAC$ 
proof–
  obtain  $lgAC$  where  $lgAC1: \text{largest-finite-antichain } A \ r \ lgAC$ 
using assms exists-largest-finite-antichain
by auto
hence  $lgAC: \text{largest-antichain } A \ r \ lgAC \wedge \text{finite } lgAC$ 
using largest-finite-antichain-def
by (metis bot-nat-0.extremum card-eq-0-iff largest-antichain-def)
then obtain  $sCD$  where  $sCD: (\text{chain-decomposition } A \ r \ sCD \wedge \text{finite } sCD) \wedge$ 
 $\text{card } sCD \leq \text{card } lgAC$ 
by (meson Dilworth-countable-aux)
then obtain  $C$  where  $C: \text{finite-smallest-chain-decomposition } A \ r \ C$ 
using exists-smallest-finite-chain-decomposition finite-smallest-chain-decomposition-def

  by blast
hence  $\text{card } lgAC \leq \text{card } C$ 
using  $lgAC \ sCD \text{ antichain-le-chain-decomposition[of } C \ lgAC]$ 
  largest-antichain-def finite-smallest-chain-decomposition-def
by blast
moreover
have  $\text{card } C \leq \text{card } lgAC$  using  $C \text{ finite-smallest-chain-decomposition-def } sCD$ 
using le-trans by blast
ultimately
have  $\text{card } lgAC = \text{card } C$  by auto

```

thus *?thesis* using *C lgAC1*
 by *metis*
 qed
 end

References

- [1] Alonzo Church. *Introduction to Mathematical Logic*, volume 1. Princeton University Press, Princeton, New Jersey, 1956.
- [2] Nicolaas Govert De Bruijn and Pál Erdős. A Colour Problem for Infinite Graphs and a Problem in the Theory of Relations. *Indagationes Mathematicae (Proceedings)*, 54:371–373, 1951.
- [3] Robert P. Dilworth. A Decomposition Theorem for Partially Ordered Sets. *Annals of Mathematics*, 51(1):161–166, 1950. <https://doi.org/10.2307/1969503>.
- [4] Philip Hall. On representatives of subsets. *London Mathematical Society*, 10:26–30, 1935. <https://doi.org/10.1112/jlms/s1-10.37.26>.
- [5] Dongchen Jiang and Tobias Nipkow. Hall’s Marriage Theorem. *Archive of Formal Proofs*, December 2010. https://www.isa-afp.org/browser_info/current/AFP/Marriage/document.pdf.
- [6] Dongchen Jiang and Tobias Nipkow. Proof Pearl: The Marriage Theorem. In *Proceedings First International Conference on Certified Programs and Proofs - CPP*, volume 7086 of *LNCs*, pages 394–399, 2011. https://doi.org/10.1007/978-3-642-25379-9_28.
- [7] Vivek Soorya Maadoori, Syed Mohammad Meesum, Shiv Pillai, T. V. H. Prathamesh, and Aditya Swami. Formal Proof of Dilworth’s Theorem. *Archive of Formal Proofs*, March 2025. <https://isa-afp.org/entries/Dilworth.html>, Formal proof development.
- [8] Eric Charles Milner. Dilworth’s Decomposition Theorem in the Infinite Case. In Kenneth P. Bogart, Ralph Freese, and Joseph P. S. Kung, editors, *The Dilworth Theorems: Selected Papers of Robert P. Dilworth*, Contemporary Mathematicians. Birkhäuser, 1990.
- [9] Micha A. Perles. On dilworth’s theorem in the infinite case. *Israel Journal of Mathematics*, 1(2):108–109, 1963.
- [10] Richard Rado. Axiomatic treatment of rank in infinite sets. *Canadian Journal of Mathematics*, 1(4):337–343, 1949.

- [11] Fabián Fernando Serrano Suárez. *Formalización en Isar de la Meta-Lógica de Primer Orden*. PhD thesis, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla, Spain, 2012. <https://idus.us.es/handle/11441/57780>. In Spanish.
- [12] Fabián Fernando Serrano Suárez, Mauricio Ayala-Rincón, and Thaynara Arielly de Lima. Hall’s Theorem for Enumerable Families of Finite Sets. In *Proc. Int. Conf. on Intelligent Computer Mathematics CICM*, volume 13467 of *LNCS*, pages 107–121. Springer, 2022. https://doi.org/10.1007/978-3-031-16681-5_7.
- [13] Fabián Fernando Serrano Suárez, Mauricio Ayala-Rincón, and Thaynara Arielly de Lima. Formalisation of Hall’s Theorem for Countable Infinite Graphs. In *Proc. Colombian Congress on Computation 18-CCC*, volume 2208 of *Communications in Computer and Information Science - Advances in Computing*, Cham, 2024. Springer. https://doi.org/10.1007/978-3-031-75233-9_15.
- [14] Abhishek Kr Singh. Fully Mechanized Proofs of Dilworths Theorem and Mirskys Theorem, 2017. <https://arxiv.org/abs/1703.06133>.
- [15] Fabián Fernando Serrano Suárez, Thaynara Arielly de Lima, and Mauricio Ayala-Rincón. Compactness Theorem for Propositional Logic and Combinatorial Applications. *Arch. Formal Proofs*, 2024. https://www.isa-afp.org/entries/Prop_Compactness.html.