Formal Proof of Dilworth's Theorem

Vivek Soorya Maadoori   S. M. Meesum   Shiv Pillai
T. V. H. Prathamesh   Aditya Swami

March 25, 2025

Abstract

A *chain* is defined as a totally ordered subset of a partially ordered set. A *chain cover* refers to a collection of chains of a partially ordered set whose union equals the entire set. A *chain decomposition* is a chain cover consisting of pairwise disjoint sets. An *antichain* is a subset of elements of a partially ordered set in which no two elements are comparable.

In 1950, Dilworth proved that in any finite partially ordered set, the cardinality of a largest antichain equals the cardinality of a smallest chain decomposition.[2]

In this paper, we formalise a proof of the theorem above, also known as *Dilworth's theorem*, based on a proof by Perles (1963) [3]. Our formalisation draws on the formalisation of Dilworth's theorem for chain covers in Coq by Abhishek Kr. Singh [4], and builds on the AFP entry containing formalisation of minimal and maximal elements in a set by Martin Desharnais [1]. Our formalisation extends the prior work in Coq by including a formal proof of Dilworth's theorem for chain decomposition.

Contents

theory *Dilworth*
imports *Main HOL.Complete-Partial-Order HOL.Relation HOL.Order-Relation Min-Max-Least-Greatest.Min-Max-Least-Greatest-Set*
begin

Note: The Dilworth's theorem for chain cover is labelled Dilworth and the extension to chain decomposition is labelled Dilworth_Decomposition.

context *order*
begin

1 Definitions

definition *chain-on* :: - *set* ⇒ - *set* ⇒ *bool* **where**
$chain\text{-}on\ A\ S \longleftrightarrow ((A \subseteq S) \wedge (Complete\text{-}Partial\text{-}Order.chain\ (\leq)\ A))$

definition *antichain* :: - *set* ⇒ *bool* **where**
$antichain\ S \longleftrightarrow (\forall x \in S.\ \forall y \in S.\ (x \leq y \vee y \leq x) \longrightarrow x = y)$

definition *antichain-on* :: - *set* ⇒ - *set* ⇒ *bool* **where**
$(antichain\text{-}on\ A\ S) \longleftrightarrow$
$(partial\text{-}order\text{-}on\ A\ (relation\text{-}of\ (\leq)\ A)) \wedge (S \subseteq A) \wedge (antichain\ S)$

definition *largest-antichain-on*:: - *set* ⇒ - *set* ⇒ *bool* **where**
$largest\text{-}antichain\text{-}on\ P\ lac \longleftrightarrow$
$(antichain\text{-}on\ P\ lac \wedge (\forall\ ac.\ antichain\text{-}on\ P\ ac \longrightarrow card\ ac \leq card\ lac))$

definition *chain-cover-on*:: - *set* ⇒ - *set set* ⇒ *bool* **where**
$chain\text{-}cover\text{-}on\ S\ cv \longleftrightarrow (\bigcup\ cv = S) \wedge (\forall\ x \in cv\ .\ chain\text{-}on\ x\ S)$

definition *antichain-cover-on*:: - *set* ⇒ - *set set* ⇒ *bool* **where**
$antichain\text{-}cover\text{-}on\ S\ cv \longleftrightarrow (\bigcup\ cv = S) \wedge (\forall\ x \in cv\ .\ antichain\text{-}on\ S\ x)$

definition *smallest-chain-cover-on*:: - *set* ⇒ - *set set* ⇒ *bool* **where**
$smallest\text{-}chain\text{-}cover\text{-}on\ S\ cv \equiv$
$(chain\text{-}cover\text{-}on\ S\ cv\ \wedge$
$(\forall\ cv2.\ (chain\text{-}cover\text{-}on\ S\ cv2 \wedge card\ cv2 \leq card\ cv) \longrightarrow card\ cv = card\ cv2))$

definition *chain-decomposition* **where**
$chain\text{-}decomposition\ S\ cd \equiv ((chain\text{-}cover\text{-}on\ S\ cd)\ \wedge$
$(\forall\ x \in cd.\ \forall\ y \in cd.\ x \neq y \longrightarrow (x \cap y = \{\})))$

```
definition smallest-chain-decomposition:: - set ⇒ - set set ⇒ bool where
smallest-chain-decomposition S cd
  ≡ (chain-decomposition S cd
     ∧ (∀ cd2. (chain-decomposition S cd2 ∧ card cd2 ≤ card cd)
                                         ⟶ card cd = card cd2))
```

2 Preliminary Lemmas

The following lemma shows that given a chain and an antichain, if the cardinality of their intersection is equal to 0, then their intersection is empty..

```
lemma inter-nInf:
  assumes a1: Complete-Partial-Order.chain (⊆) X
      and a2: antichain Y
  and asmInf: card (X ∩ Y) = 0
        shows X ∩ Y = {}
proof (rule ccontr)
  assume X ∩ Y ≠ {}
  then obtain a b where 1:a ∈ (X ∩ Y) b ∈ (X ∩ Y) using asmInf by blast
  then have in-chain: a ∈ X ∧ b ∈ X using 1 by simp
  then have 3: (a ≤ b) ∨ (b ≤ a) using a1
    by (simp add: chain-def)
  have in-antichain: a ∈ Y ∧ b ∈ Y using 1 by blast
  then have a = b using antichain-def a2 3
    by (metis order-class.antichain-def)
  then have ∀ a ∈ (X ∩ Y). ∀ b ∈ (X ∩ Y). a = b
    using 1 a1 a2 order-class.antichain-def
    by (smt (verit, best) IntE chain-def)
  then have card (X ∩ Y) = 1 using 1 a1 a2 card-def
    by (smt (verit, best) all-not-in-conv asmInf card-0-eq card-le-Suc0-iff-eq
        finite-if-finite-subsets-card-bdd subset-eq subset-iff)
  then show False using asmInf by presburger
qed
```

The following lemma shows that given a chain X and an antichain Y that both are subsets of S, their intersection is either empty or has cardinality one..

```
lemma chain-antichain-inter:
  assumes a1: Complete-Partial-Order.chain (⊆) X
      and a2: antichain Y
      and a3: X ⊆ S ∧ Y ⊆ S
        shows (card (X ∩ Y) = 1) ∨ ((X ∩ Y) = {})
proof (cases card (X ∩ Y) ≥ 1)
  case True
  then obtain a b where 1:a ∈ (X ∩ Y) b ∈ (X ∩ Y)
    by (metis card-1-singletonE insert-subset obtain-subset-with-card-n)
  then have a ∈ X ∧ b ∈ X using 1 by blast
  then have 3: (a ≤ b) ∨ (b ≤ a) using Complete-Partial-Order.chain-def a1
```

```
    by (smt (verit, best))
  have a ∈ Y ∧ b ∈ Y using 1 by blast
  then have a = b using a2 order-class.antichain-def 3
    by (metis)
  then have ∀ a ∈ (X ∩ Y). ∀ b ∈ (X ∩ Y). a = b
    using 1 a1 a2 order-class.antichain-def
    by (smt (verit, best) Int-iff chainD)
  then have card (X ∩ Y) = 1 using 1 a1 a2
    by (metis One-nat-def True card.infinite card-le-Suc0-iff-eq
              order-class.order-antisym zero-less-one-class.zero-le-one)
  then show ?thesis by presburger
next
  case False
  then have card (X ∩ Y) < 1 by linarith
  then have card (X ∩ Y) = 0 by blast
  then have X ∩ Y = {} using assms inter-nInf by blast
  then show ?thesis by force
qed
```

Following lemmas show that given a finite set S, there exists a chain decomposition of S.

```
lemma po-restr: assumes partial-order-on B r
                    and A ⊆ B
                  shows partial-order-on A (r ∩ (A × A))
  using assms
  unfolding partial-order-on-def preorder-on-def antisym-def refl-on-def trans-def
 by (metis (no-types, lifting) IntD1 IntD2 IntI Int-lower2 inf.orderE mem-Sigma-iff)


lemma eq-restr: (Restr (relation-of (≤) (insert a A)) A) = (relation-of (≤) A)
  (is ?P = ?Q)
proof
  show ?P ⊆ ?Q
  proof
    fix z
    assume z ∈ ?P
    then obtain x y where tuple: (x, y) = z using relation-of-def by blast
    then have 1: (x, y) ∈ ((relation-of (≤) (insert a A)) ∩ (A × A))
      using relation-of-def
      using ‹z ∈ Restr (relation-of (≤) (insert a A)) A› by blast
    then have 2: (x, y) ∈ (relation-of (≤) (insert a A)) by simp
    then have 3: (x, y) ∈ (A × A) using 1 by simp
    then have (x, y) ∈ (A × A) ∧ (x ≤ y) using relation-of-def 2
      by (metis (no-types, lifting) case-prodD mem-Collect-eq)
    then have (x, y) ∈ (relation-of (≤) A) using relation-of-def by blast
    then show z ∈ ?Q using tuple by fast
  qed
next
  show ?Q ⊆ ?P
```

```
  proof
    fix z
    assume asm1: z ∈ ?Q
    then obtain x y where tuple: (x, y) = z by (metis prod.collapse)
    then have 0: (x, y) ∈ (A × A) ∧ (x ≤ y) using asm1 relation-of-def
      by (metis (mono-tags, lifting) case-prod-conv mem-Collect-eq)
    then have 1: (x, y) ∈ (A × A) by fast
    have rel: x ≤ y using 0 by blast
    have (A × A) ⊆ ((insert a A) × (insert a A)) by blast
    then have (x, y) ∈ ((insert a A) × (insert a A)) using 1 by blast
    then have (x, y) ∈ (relation-of (≤) (insert a A))
      using rel relation-of-def by blast
   then have (x, y) ∈ ((relation-of (≤) (insert a A)) ∩ (A × A)) using 1 by fast
    then show z ∈ ?P using tuple by fast
  qed
qed

lemma part-ord:partial-order-on S (relation-of (≤) S)
  by (smt (verit, ccfv-SIG) local.dual-order.eq-iff local.dual-order.trans
      partial-order-on-relation-ofI)
```

The following lemma shows that a chain decomposition exists for any finite set S.

```
lemma exists-cd: assumes finite S
                 shows ∃ cd. chain-decomposition S cd
   using assms
proof(induction rule: finite.induct)
  case emptyI
 then show ?case using assms unfolding chain-decomposition-def chain-cover-on-def
    by (metis Sup-empty empty-iff)
next
  case (insertI A a)
  show ?case using assms
  proof (cases a ∈ A)
    case True
    then have 1: (insert a A) = A by fast
    then have ∃ X. chain-decomposition A X using insertI by simp
    then show ?thesis using 1 by auto
  next
    case False
    have subset-a: {a} ⊆ (insert a A) by simp
    have chain-a: Complete-Partial-Order.chain (≤) {a}
      using chain-singleton chain-def by auto
    have subset-A: A ⊆ (insert a A) by blast
    have partial-a: partial-order-on A ((relation-of (≤) (insert a A)) ∩ (A × A))
      using po-restr insertI subset-A part-ord by blast
    then have chain-on-A: chain-on {a} (insert a A)
      unfolding order-class.chain-on-def using chain-a partial-a
              insertI.prems chain-on-def by simp
```

then obtain *X* **where** *chain-set*: *chain-decomposition A X*
  using *insertI partial-a eq-restr*
  by *auto*
have *chains-X*: $\forall\ x \in (insert\ \{a\}\ X).\ chain\text{-}on\ x\ (insert\ a\ A)$
  using *subset-A chain-set chain-on-def*
    chain-decomposition-def chain-cover-on-def chain-on-A
  by *auto*
have *subsets-X*: $\forall\ x \in (insert\ \{a\}\ X).\ x \subseteq (insert\ a\ A)$
  using *chain-set chain-decomposition-def subset-a chain-cover-on-def*
  by *auto*
have *null-inter-X*: $\forall\ x \in X.\ \forall\ y \in X.\ x \neq y \longrightarrow x \cap y = \{\}$
  using *chain-set chain-decomposition-def*
  by (*simp add*: *order-class.chain-decomposition-def*)
have $\{a\} \notin X$ **using** *False chain-set chain-decomposition-def chain-cover-on-def*
  by (*metis UnionI insertCI*)
then have *null-inter-a*: $\forall\ x \in X.\ \{a\} \cap x = \{\}$
  using *False chain-set order-class.chain-decomposition-def*
  using *chain-decomposition-def chain-cover-on-def* **by** *auto*
then have *null-inter*: $\forall\ x \in (insert\ \{a\}\ X).\ \forall\ y \in (insert\ \{a\}\ X).\ x \neq y \longrightarrow x \cap y = \{\}$
  using *null-inter-X* **by** *simp*
have *union*: $\bigcup\ (insert\ \{a\}\ X) = (insert\ a\ A)$ **using** *chain-set*
  by (*simp add*: *chain-decomposition-def chain-cover-on-def*)
have *chain-decomposition* (*insert a A*) (*insert* {*a*} *X*)
  using *subsets-X chains-X union null-inter* **unfolding** *chain-decomposition-def*

    chain-cover-on-def
  by *simp*
then show *?thesis* **by** *blast*
qed
qed

The following lemma shows that the chain decomposition of a set is a chain cover.

lemma *cd-cv*:
  assumes *chain-decomposition P cd*
  shows *chain-cover-on P cd*
  using *assms* **unfolding** *chain-decomposition-def* **by** *argo*

The following lemma shows that for any finite partially ordered set, there exists a chain cover on that set.

lemma *exists-chain-cover*: **assumes** *finite P*
    shows $\exists\ cv.\ chain\text{-}cover\text{-}on\ P\ cv$
proof−
  show *?thesis* **using** *assms exists-cd cd-cv* **by** *blast*
qed

lemma *finite-cv-set*: **assumes** *finite P*
    and $S = \{x.\ chain\text{-}cover\text{-}on\ P\ x\}$

shows *finite S*
proof−
have *1*: ∀ *cv. chain-cover-on P cv* ⟶ (∀ *c* ∈ *cv. finite c*)
unfolding *chain-cover-on-def chain-on-def chain-def*
using *assms*(*1*) *rev-finite-subset* **by** *auto*
have *2*: ∀ *cv. chain-cover-on P cv* ⟶ *finite cv*
unfolding *chain-cover-on-def*
using *assms*(*1*) *finite-UnionD* **by** *auto*
have ∀ *cv. chain-cover-on P cv* ⟶ (∀ *c* ∈ *cv. c* ⊆ *P*)
unfolding *chain-cover-on-def* **by** *blast*
then have ∀ *cv. chain-cover-on P cv* ⟶ *cv* ⊆ *Fpow P* **using** *Fpow-def 1* **by** *fast*
then have ∀ *cv. chain-cover-on P cv* ⟶ *cv* ∈ *Fpow* (*Fpow P*)
using *Fpow-def 2* **by** *fast*
then have *S* ⊆ *Fpow* (*Fpow P*) **using** *assms*(*2*) **by** *blast*
then show *?thesis*
using *assms*(*1*) **by** (*meson Fpow-subset-Pow finite-Pow-iff finite-subset*)
qed

The following lemma shows that for every element of an antichain in a set, there exists a chain in the chain cover of that set, such that the element of the antichain belongs to the chain.

lemma *elem-ac-in-c*: **assumes** *a1*: *antichain-on P ac*
and *chain-cover-on P cv*
shows ∀ *a* ∈ *ac.* ∃ *c* ∈ *cv. a* ∈ *c*
proof−
have ⋃ *cv* = *P* **using** *assms*(*2*) *chain-cover-on-def* **by** *simp*
then have *ac* ⊆ ⋃ *cv* **using** *a1 antichain-on-def* **by** *simp*
then show ∀ *a* ∈ *ac.* ∃ *c* ∈ *cv. a* ∈ *c* **by** *blast*
qed

For a function f that maps every element of an antichain to some chain it belongs to in a chain cover, we show that, the co-domain of f is a subset of the chain cover.

lemma *f-image*: **fixes** *f* :: - ⇒ - *set*
assumes *a1*: (*antichain-on P ac*)
and *a2*: (*chain-cover-on P cv*)
and *a3*: ∀ *a* ∈ *ac.* ∃ *c* ∈ *cv. a* ∈ *c* ∧ *f a* = *c*
shows (*f* ‘ *ac*) ⊆ *cv*
proof
have *1*: ∀ *a* ∈ *ac.* ∃ *c* ∈ *cv. a* ∈ *c* **using** *elem-ac-in-c a1 a2* **by** *presburger*
fix *y*
assume *y* ∈ (*f* ‘ *ac*)
then obtain *x* **where** *f x* = *y* *x* ∈ *ac* **using** *a1 a2* **by** *auto*
then have *x* ∈ *y* **using** *a3* **by** *blast*
then show *y* ∈ *cv* **using** *a3* **using** ‹*f x* = *y*› ‹*x* ∈ *ac*› **by** *blast*
qed

3 Size of an antichain is less than or equal to the size of a chain cover

The following lemma shows that given an antichain ac and chain cover cv on a finite set, the cardinality of ac will be less than or equal to the cardinality of cv.

```
lemma antichain-card-leq:
          assumes (antichain-on P ac)
              and (chain-cover-on P  cv)
              and finite P
            shows card ac ≤ card cv
proof (rule ccontr)
  assume a-contr: ¬ card ac ≤ card cv
  then have 1: card cv < card ac by simp
  have finite-cv: finite cv using assms(2,3) chain-cover-on-def
    by (simp add: finite-UnionD)
  have 2: ∀ a ∈ ac. ∃ c ∈ cv. a ∈ c using assms(1,2) elem-ac-in-c by simp
  then obtain f where f-def: ∀ a ∈ ac. ∃ c ∈ cv. a ∈ c ∧ f a = c by metis
  then have (f ‘ ac) ⊆ cv using f-image assms by blast
  then have 3: card (f ‘ ac) ≤ card cv using f-def finite-cv card-mono by metis
  then have card (f ‘ ac) < card ac using 1 by auto
  then have ¬ inj-on f ac using pigeonhole by blast
  then obtain a b where p1: f a = f b a ≠ b a ∈ ac b ∈ ac
    using inj-def f-def by (meson inj-on-def)
  then have antichain-elem: a ∈ ac ∧ b ∈ ac using f-def by blast
  then have ∃ c ∈ cv. f a = c ∧ f b = c using f-def 2 1 ‹f ‘ ac ⊆ cv› p1(1) by
auto
  then have chain-elem: ∃ c ∈ cv. a ∈ c ∧ b ∈ c
    using f-def p1(1) p1(3) p1(4) by blast
  then have a ≤ b ∨ b ≤ a using chain-elem chain-cover-on-def chain-on-def
    by (metis assms(2) chainD)
  then have a = b
    using antichain-elem assms(1) antichain-on-def antichain-def by auto
  then show False using p1(2) by blast
qed
```

4 Existence of a chain cover whose cardinality is the cardinality of the largest antichain

4.1 Preliminary lemmas

The following lemma shows that the maximal set is an antichain.

```
lemma maxset-ac: antichain ({x . is-maximal-in-set P x})
  using antichain-def local.is-maximal-in-set-iff by auto
```

The following lemma shows that the minimal set is an antichain.

lemma *minset-ac*: *antichain* $(\{x\ .\ \textit{is-minimal-in-set}\ P\ x\})$
  using *antichain-def is-minimal-in-set-iff* **by** *force*

The following lemma shows that the null set is both an antichain and a chain cover.

lemma *antichain-null*: *antichain* $\{\}$
proof−
  show *?thesis* **using** *antichain-def* **by** *simp*
qed

lemma *chain-cover-null*: **assumes** $P = \{\}$ **shows** *chain-cover-on* P $\{\}$
proof−
  show *?thesis* **using** *chain-cover-on-def*
    by (*simp add*: *assms*)
qed

The following lemma shows that for any arbitrary x that does not belong to the largest antichain of a set, there exists an element y in the antichain such that x is related to y or y is related to x.

lemma *x-not-in-ac-rel*: **assumes** *largest-antichain-on* P *ac*
    and $x \in P$
    and $x \notin ac$
    and *finite* P
  shows $\exists\ y \in ac.\ (x \leq y) \vee (y \leq x)$
proof (*rule ccontr*)
  assume $\neg\ (\exists\, y{\in}ac.\ x \leq y \vee y \leq x)$
  then have *1*: $\forall\ y \in ac.\ (\neg(x \leq y) \wedge \neg(y \leq x))$ **by** *simp*
  then have *2*: $\forall\ y \in ac.\ x \neq y$ **by** *auto*
  then obtain S **where** *S-def*: $S = \{x\} \cup ac$ **by** *blast*
  then have *S-fin*: *finite* S
    using *assms*(*4*) *assms*(*1*) *assms*(*2*) *largest-antichain-on-def antichain-on-def*
    by (*meson Un-least bot.extremum insert-subset rev-finite-subset*)
  have *S-on-P*: *antichain-on* P S
   using *S-def largest-antichain-on-def antichain-on-def assms*(*1*,*2*) *1 2 antichain-def*

    by *auto*
  then have $ac \subset S$ **using** *S-def assms*(*3*) **by** *auto*
  then have *card ac* $<$ *card S* **using** *psubset-card-mono S-fin* **by** *blast*
  then show *False* **using** *assms*(*1*) *largest-antichain-on-def S-on-P* **by** *fastforce*
qed

The following lemma shows that for any subset Q of the partially ordered P, if the minimal set of P is a subset of Q, then it is a subset of the minimal set of Q as well.

lemma *minset-subset-minset*:
  assumes *finite* P
    and $Q \subseteq P$
    and $\forall\ x.\ (\textit{is-minimal-in-set}\ P\ x \longrightarrow x \in Q)$

```
  shows {x . is-minimal-in-set P x} ⊆ {x. is-minimal-in-set Q x}
proof
 fix x
 assume asm1: x ∈ {z. is-minimal-in-set P z}
 have 1: x ∈ Q using asm1 assms(3)
  by blast
 have partial-Q: partial-order-on Q (relation-of (≤) Q)
  using assms(1) assms(3) partial-order-on-def
  by (simp add: partial-order-on-relation-ofI)
 have ∀ q ∈ Q. q ∈ P using assms(2) by blast
 then have is-minimal-in-set Q x using is-minimal-in-set-iff 1 partial-Q
  using asm1 by force
 then show x ∈ {z. is-minimal-in-set Q z} by blast
qed
```

The following lemma show that if P is not empty, the minimal set of P is not empty.

```
lemma non-empty-minset: assumes finite P
                            and P ≠ {}
                          shows {x . is-minimal-in-set P x} ≠ {}
 by (simp add: assms ex-minimal-in-set)
```

The following lemma shows that for all elements m of the minimal set, there exists a chain c in the chain cover such that m belongs to c.

```
lemma elem-minset-in-chain: assumes finite P
                                and chain-cover-on P cv
                              shows is-minimal-in-set P a ⟶ (∃ c ∈ cv. a ∈ c)
 using assms(2) chain-cover-on-def is-minimal-in-set-iff by auto
```

The following lemma shows that for all elements m of the maximal set, there exists a chain c in the chain cover such that m belongs to c.

```
lemma elem-maxset-in-chain: assumes finite P
                                and chain-cover-on P  cv
                              shows is-maximal-in-set P a ⟶ (∃ c ∈ cv. a ∈ c)
 using chain-cover-on-def assms is-maximal-in-set-iff by auto
```

The following lemma shows that for a given chain cover and antichain on P, if the cardinality of the chain cover is equal to the cardinality of the antichain then for all chains c of the chain cover, there exists an element a of the antichain such that a belongs to c.

```
lemma card-ac-cv-eq: assumes finite P
                         and chain-cover-on P cv
                         and antichain-on P  ac
                         and card cv = card ac
                       shows ∀ c ∈ cv. ∃ a ∈ ac. a ∈ c
proof (rule ccontr)
 assume ¬ (∀ c∈cv. ∃ a∈ac. a ∈ c)
 then obtain c where c ∈ cv ∀ a ∈ ac. a ∉ c by blast
```

```
  then have ∀ a ∈ ac. a ∈ ⋃ (cv − {c}) (is ∀ a ∈ ac. a ∈ ?cv-c)
    using assms(2,3) unfolding chain-cover-on-def antichain-on-def by blast
  then have 1: ac ⊆ ?cv-c by blast
  have 2: partial-order-on ?cv-c (relation-of (≤) ?cv-c)
    using assms(1) assms(3) partial-order-on-def
    by (simp add: partial-order-on-relation-ofI)
  then have ac-on-cv-v: antichain-on ?cv-c ac
    using 1 assms(3) antichain-on-def unfolding antichain-on-def by blast
  have 3: ∀ a ∈ (cv − {c}). a ⊆ ?cv-c by auto
  have 4: ∀ a ∈ (cv − {c}). Complete-Partial-Order.chain (≤) a using assms(2)

    unfolding chain-cover-on-def chain-on-def
    by (meson DiffD1 Union-upper chain-subset)
  have 5: ∀ a ∈ (cv − {c}). chain-on a ?cv-c using chain-on-def 2 3 4
    by metis
  have ⋃ (cv − {c}) = ?cv-c by simp
  then have cv-on-cv-v: chain-cover-on ?cv-c (cv − {c})
    using 5 chain-cover-on-def by simp
  have card (cv − {c}) < card cv
    by (metis ‹c ∈ cv› assms(1) assms(2) card-Diff1-less
        chain-cover-on-def finite-UnionD)
  then have card (cv − {c}) < card ac using assms(4) by simp
  then show False using ac-on-cv-v cv-on-cv-v antichain-card-leq assms part-ord
   by (metis Diff-insert-absorb Diff-subset Set.set-insert Union-mono assms(2,4)
        card-Diff1-less-iff card-seteq chain-cover-on-def rev-finite-subset)
qed
```

The following lemma shows that if an element m from the minimal set is in a chain, it is less than or equal to all elements in the chain.

```
lemma e-minset-lesseq-e-chain: assumes chain-on c P
                                    and is-minimal-in-set P m
                                    and m ∈ c
                                  shows ∀ a ∈ c. m ≤ a
proof−
  have 1: c ⊆ P using assms(1) unfolding chain-on-def by simp
  then have is-minimal-in-set c m using 1 assms(2,3) is-minimal-in-set-iff by
auto

  then have 3: ∀ a ∈ c. (a ≤ m) ⟶ a = m unfolding is-minimal-in-set-iff by
auto
  have ∀ a ∈ c. ∀ b ∈ c. (a ≤ b) ∨ (b ≤ a) using assms(1)
    unfolding chain-on-def chain-def by blast
  then show ?thesis using 3 assms(3) by blast
qed
```

The following lemma shows that if an element m from the maximal set is in a chain, it is greater than or equal to all elements in the chain.

```
lemma e-chain-lesseq-e-maxset: assumes chain-on c P
                                    and is-maximal-in-set P m
```

and $m \in c$
shows $\forall\ a \in c.\ a \leq m$
using *assms chainE chain-on-def is-maximal-in-set-iff local.less-le-not-le subsetD*

by *metis*

The following lemma shows that for any two elements of an antichain, if they both belong to the same chain in the chain cover, they must be the same element.

lemma *ac-to-c* : **assumes** *finite P*
and *chain-cover-on P cv*
and *antichain-on P ac*
shows $\forall\ a \in ac.\ \forall\ b \in ac.\ \exists\ c \in cv.\ a \in c \land b \in c \longrightarrow a = b$
proof−
show *?thesis*
using *assms chain-cover-on-def antichain-on-def*
unfolding *chain-cover-on-def chain-on-def chain-def antichain-on-def antichain-def*

by (*meson assms*(*2*,*3*) *elem-ac-in-c subsetD*)
qed

The following lemma shows that for two finite sets, if their cardinalities are equal, then their cardinalities would remain equal after removing a single element from both sets.

lemma *card-Diff1-eq*: **assumes** *finite A*
and *finite B*
and *card A* = *card B*
shows $\forall\ a \in A.\ \forall\ b \in B.$ *card* $(A - \{a\})$ = *card* $(B - \{b\})$
proof−
show *?thesis* **using** *assms*(*3*) **by** *auto*
qed

The following lemma shows that for two finite sets A and B of equal cardinality, removing two unique elements from A and one element from B will ensure the cardinality of A is less than B.

lemma *card-Diff2-1-less*: **assumes** *finite A*
and *finite B*
and *card A* = *card B*
and $a \in A$
and $b \in A$
and $a \neq b$
shows $\forall\ x \in B.$ *card* $((A - \{a\}) - \{b\}) <$ *card* $(B - \{x\})$
proof−
show *?thesis*
by (*metis DiffI assms card-Diff1-eq card-Diff1-less-iff finite-Diff singletonD*)
qed

The following lemma shows that for all elements of a partially ordered set, there exists an element in the minimal set that will be less than or equal to it.

lemma *min-elem-for-P*: **assumes** *finite P*
shows $\forall\ p \in P.\ \exists\ m.$ *is-minimal-in-set P m* $\wedge\ m \leq p$
proof
fix *p*
assume *asm*:$p \in P$
obtain *m* **where** *m*: $m \in P\ \ m \leq p\ \forall a \in P.\ a \leq m \longrightarrow a = m$
using *finite-has-minimal2*[*OF assms*(*1*) *asm*] **by** *metis*
hence *is-minimal-in-set P m* **unfolding** *is-minimal-in-set-iff*
using *part-ord* **by** *force*
then show $\exists m.$ *is-minimal-in-set P m* $\wedge\ m \leq p$ **using** *m*
by *blast*
qed

The following lemma shows that for all elements of a partially ordered set, there exists an element in the maximal set that will be greater than or equal to it.

lemma *max-elem-for-P*: **assumes** *finite P*
shows $\forall\ p \in P.\ \exists\ m.$ *is-maximal-in-set P m* $\wedge\ p \leq m$
using *assms finite-has-maximal2*
by (*metis dual-order.strict-implies-order is-maximal-in-set-iff*)

The following lemma shows that if the minimal set is not considered as the largest antichain on a set, then there exists an element a in the minimal set such that a does not belong to the largest antichain.

lemma *min-e-nIn-lac*: **assumes** *largest-antichain-on P ac*
and $\{x.$ *is-minimal-in-set P x*$\} \neq ac$
and *finite P*
shows $\exists m.$ (*is-minimal-in-set P m*) $\wedge$ $(m \notin ac)$
(**is** $\exists m.$ (*?ms m*) $\wedge$ $(m \notin ac)$)
proof (*rule ccontr*)
assume *asm*:$\neg$ ($\exists\ m.$ (*?ms m*) $\wedge$ $(m \notin ac)$)
then have $\forall\ m.\ \neg$(*?ms m*) $\vee\ \ m \in ac$ **by** *blast*
then have *1*: $\{m$. *?ms m*$\} \subseteq ac$ **by** *blast*
then show *False*
proof *cases*
assume $\{m$. *?ms m*$\} = ac$
then show *?thesis* **using** *assms*(*2*) **by** *blast*
next
assume $\neg$ ($\{m$. *?ms m*$\} = ac$)
then have *1*:$\{m$. *?ms m*$\} \subset ac$ **using** *1* **by** *simp*
then obtain *y* **where** *y-def*: $y \in ac$ *?ms y* **using** *asm assms*(*1*,*3*)
by (*metis chain-cover-null elem-ac-in-c empty-subsetI ex-in-conv largest-antichain-on-def local.ex-minimal-in-set psubsetE*)
then have *y-in-P*: $y \in P$
using *y-def*(*1*) *assms*(*1*) *largest-antichain-on-def antichain-on-def* **by** *blast*
then have *2*: $\forall\ x.$ (*?ms x* $\longrightarrow x \neq y$) **using** *y-def*(*2*) *1 assms*(*1*,*3*)

```
   using asm min-elem-for-P  DiffE mem-Collect-eq psubset-imp-ex-mem sub-
set-iff
      unfolding largest-antichain-on-def antichain-def antichain-on-def
      by (smt (verit))
    have partial-P: partial-order-on P (relation-of (≤) P)
      using assms(1) largest-antichain-on-def antichain-on-def by simp
    then have ∀ x. ?ms x ⟶ ¬ (y ≤ x) using 2 unfolding  is-minimal-in-set-iff
      using ‹y ∈ P›
      using 2 y-def(2) by blast
    then show False using y-def(2) by blast
  qed
qed
```

The following lemma shows that if the maximal set is not considered as the largest antichain on a set, then there exists an element a in the maximal set such that a does not belong to the largest antichain.

```
lemma max-e-nIn-lac: assumes largest-antichain-on P ac
                         and {x . is-maximal-in-set P x} ≠ ac
                         and finite P
                       shows ∃ m . is-maximal-in-set P m ∧  m ∉ ac
                        (is ∃ m. ?ms m ∧ m ∉ ac)
proof (rule ccontr)
  assume asm:¬ (∃ m. ?ms m ∧ m ∉ ac)
  then have ∀ m . ¬ ?ms m ∨ m ∈ ac by blast
  then have 1: {x . ?ms x} ⊆ ac by blast
  then show False
  proof cases
    assume asm: {x . ?ms x} = ac
    then show ?thesis using assms(2) by blast
  next
    assume ¬ ({x . ?ms x} = ac)
    then have {x . ?ms x} ⊂ ac using 1 by simp
    then obtain y where y-def: y ∈ ac ¬ (?ms y) using assms asm
      by blast
    then have y-in-P: y ∈ P
      using y-def(1) assms(1) largest-antichain-on-def antichain-on-def by blast
    then have 2: ∀ x . ?ms x  ⟶ x ≠ y using y-def(2) by auto
    have partial-P: partial-order-on P (relation-of (≤) P)
      using assms(1) largest-antichain-on-def antichain-on-def by simp
    then have ∀ x . ?ms x  ⟶ ¬ (x ≤ y) using 2 unfolding  is-maximal-in-set-iff
      using ‹y ∈ P›
      using local.dual-order.order-iff-strict by auto
    then have 3: ∀ x . ?ms x  ⟶ (x > y) ∨ ¬ (x ≤ y) by blast
    then show False
    proof cases
      assume asm1: ∃ x. ?ms x ∧ (x > y)
      have ∀ x ∈ ac. (x ≤ y) ∨ (y ≤ x) ⟶ x = y using assms(1) y-def(1)
        unfolding largest-antichain-on-def antichain-on-def antichain-def by simp
      then have ∀ x . ?ms x  ⟶ (x > y) ⟶ x = y using 1 by auto
```

then have ∃ *x*. *?ms x* ∧ *y* = *x* **using** *asm1* **by** *auto*
then show *?thesis* **using** *2* **by** *blast*
next
assume ¬ (∃ *x*. *?ms x* ∧ (*x* > *y*))
then have ∀ *x*. *?ms x* ⟶ ¬ (*x* ≤ *y*) **using** *3* **by** *simp*
have *a*: ∃ *z* . *?ms z* ∧ *y* ≤ *z*
using *max-elem-for-P*[*OF assms*(*3*)] *y-in-P* *partial-P*
by *fastforce*

have ∀ *a*. *?ms a* ⟶ (*a* ≤ *y*) ∨ (*y* ≤ *a*) ⟶ *a* = *y* **using** *assms*(*1*) *y-def*(*1*)
1
unfolding *largest-antichain-on-def antichain-on-def antichain-def* **by** *blast*
then have ∃ *z* .*?ms z* ∧ *z* = *y* **using** *a* **by** *blast*
then show *?thesis* **using** *2* **by** *blast*
qed
qed
qed

4.2 Statement and Proof

Proves theorem for the empty set.

lemma *largest-antichain-card-eq-empty*:
assumes *largest-antichain-on P lac*
and *P* = {}
shows ∃ *cv*. (*chain-cover-on P cv*) ∧ (*card cv* = *card lac*)
proof−
have *lac* = {} **using** *assms*(*1*) *assms*(*2*)
unfolding *largest-antichain-on-def antichain-on-def* **by** *simp*
then show *?thesis* **using** *assms*(*2*) *chain-cover-null* **by** *auto*
qed

Proves theorem for the non-empty set.

lemma *largest-antichard-card-eq*:
assumes *asm1*: *largest-antichain-on P lac*
and *asm2*: *finite P*
and *asm3*: *P* ≠ {}
shows ∃ *cv*. (*chain-cover-on P cv*) ∧ (*card cv* = *card lac*)
using *assms*
— Proof by induction on the cardinality of P
proof (*induction card P arbitrary*: *P lac rule*: *less-induct*)
case *less*
let *?max* = {*x* . *is-maximal-in-set P x*}
let *?min* = {*x* . *is-minimal-in-set P x*}
have *partial-P*: *partial-order-on P* (*relation-of* (≤) *P*)
using *assms partial-order-on-def antichain-on-def largest-antichain-on-def*
less.prems(*1*) **by** *presburger*
show *?case* — the largest antichain is not the maximal set or the minimal set
proof (*cases* ∃ *ac*. (*antichain-on P ac* ∧ *ac* ≠ *?min* ∧ *ac* ≠ *?max*) ∧ *card ac* =
card lac)

case *True*
obtain *ac* **where** *ac*:*antichain-on P ac ac* $\neq$ *?min ac* $\neq$ *?max card ac* = *card lac*
using *True* **by** *force*
then have *largest-antichain-on P ac* **using** *asm1 largest-antichain-on-def*
using *less.prems*(*1*) **by** *presburger*
then have *lac-in-P*: *lac* $\subseteq$ *P*
using *asm1 antichain-on-def largest-antichain-on-def less.prems*(*1*) **by** *presburger*
then have *ac-in-P*: *ac* $\subseteq$ *P*
using *ac*(*1*) *antichain-on-def* **by** *blast*
define *p-plus* **where** *p-plus* = $\{x.\ x \in P \land (\exists\ y \in ac.\ \ y \leq x)\}$
— set of all elements greater than or equal to any given element in the largest antichain
define *p-minus* **where** *p-minus* = $\{x.\ x \in P \land (\exists\ y \in ac.\ \ x\ \leq y)\}$
— set of all elements less than or equal to any given element in the largest antichain
have *1*: *ac* $\subseteq$ *p-plus*
— Shows that the largest antichain is a subset of p plus
unfolding *p-plus-def*
proof
fix *x*
assume *a1*: $x \in ac$
then have *a2*: $x \in P$
using *asm1 largest-antichain-on-def antichain-on-def less.prems*(*1*) *ac* **by** *blast*
then have $x \leq x$ **using** *antichain-def* **by** *auto*
then show $x \in \{x \in P.\ \exists\ y \in ac.\ y \leq x\}$ **using** *a1 a2* **by** *auto*
qed
have *2*: *ac* $\subseteq$ *p-minus*
— Shows that the largest antichain is a subset of p min
unfolding *p-minus-def*
proof
fix *x*
assume *a1*: $x \in ac$
then have *a2*: $x \in P$
using *asm1 largest-antichain-on-def antichain-on-def less.prems*(*1*) *ac* **by** *blast*
then have $x \leq x$ **using** *antichain-def* **by** *auto*
then show $x \in \{x \in P.\ \exists\ y \in ac.\ x \leq y\}$ **using** *a1 a2* **by** *auto*
qed
have *lac-subset*: *ac* $\subseteq$ (*p-plus* $\cap$ *p-minus*) **using** *1 2* **by** *simp*
have *subset-lac*: (*p-plus* $\cap$ *p-minus*) $\subseteq$ *ac*
proof
fix *x*
assume $x \in$ (*p-plus* $\cap$ *p-minus*)
then obtain *a b* **where** *antichain-elems*: $a \in ac\ b \in ac\ a \leq x\ x \leq b$
using *p-plus-def p-minus-def* **by** *auto*
then have $a \leq b$ **by** *simp*

```
  then have a = b
    using antichain-elems(1) antichain-elems(2) less.prems
      asm1 largest-antichain-on-def antichain-on-def antichain-def ac by metis
  then have (a ≤ x) ∧ (x ≤ a)
    using antichain-elems(3) antichain-elems(4) by blast
  then have x = a by fastforce
  then show x ∈ ac using antichain-elems(1) by simp
qed
then have lac-pset-eq: ac = (p-plus ∩ p-minus) using lac-subset by simp
have P-PP-PM: (p-plus ∪ p-minus) = P
proof
  show (p-plus ∪ p-minus) ⊆ P
  proof
    fix x
    assume x ∈ (p-plus ∪ p-minus)
    then have x ∈ p-plus ∨ x ∈ p-minus by simp
    then have x ∈ P using p-plus-def p-minus-def by auto
    then show x ∈ P .
  qed
next
  show P ⊆ (p-plus ∪ p-minus)
  proof
    fix x
    assume x-in: x ∈ P
    then have x ∈ ac ∨ x ∉ ac by simp
    then have x ∈ (p-plus ∪ p-minus)
    proof (cases x ∈ ac)
      case True
      then show ?thesis using lac-subset by blast
    next
      case False
      then obtain y where y ∈ ac (x ≤ y) ∨ (y ≤ x)
        using asm1 False x-in  asm2
         less.prems(1) less.prems(2)
         ‹largest-antichain-on P ac› x-in x-not-in-ac-rel by blast
      then have (x ∈ p-plus) ∨ (x ∈ p-minus)
        unfolding p-plus-def p-minus-def using x-in by auto
      then show ?thesis by simp
    qed
    then show x ∈ p-plus ∪ p-minus by simp
  qed
qed
obtain a where a-def: a ∈ ?min a ∉ ac
  using asm1 ac True asm3 less.prems(1) less.prems(2) min-e-nIn-lac
  by (metis ‹largest-antichain-on P ac› mem-Collect-eq)
then have ∀ x ∈ ac. ¬ (x ≤ a)
  unfolding is-minimal-in-set-iff using partial-P lac-in-P
  using ac(1) antichain-on-def
  using local.nless-le by auto
```

then have *a-not-in-PP*: *a* $\notin$ *p-plus* **using** *p-plus-def* **by** *simp*
have *a* $\in$ *P* **using** *a-def*
  by (*simp add*: *local.is-minimal-in-set-iff*)
then have *ppl*: *card p-plus* $<$ *card P* **using** *P-PP-PM a-not-in-PP*
  by (*metis Un-upper1 card-mono card-subset-eq less.prems*(*2*)
    order-le-imp-less-or-eq)
have *p-plus-subset*: *p-plus* $\subseteq$ *P* **using** *p-plus-def* **by** *simp*
have *antichain-lac*: *antichain ac*
  using *assms*(*1*) *less.prems ac*
  unfolding *largest-antichain-on-def antichain-on-def* **by** *simp*
have *finite-PP*: *finite p-plus* **using** *asm3 p-plus-subset finite-def*
  using *less.prems*(*2*) *rev-finite-subset* **by** *blast*
have *finite-lac*: *finite ac* **using** *ac-in-P asm3 finite-def*
  using *finite-subset less.prems*(*2*) *ac* **by** *auto*
have *partial-PP*: *partial-order-on p-plus* (*relation-of* ($\leq$) *p-plus*)
  using *partial-P p-plus-subset partial-order-on-def*
  by (*smt* (*verit*, *best*) *local.antisym-conv local.le-less local.order-trans*
    partial-order-on-relation-ofI)
then have *lac-on-PP*: *antichain-on p-plus ac*
  using *antichain-on-def 1 antichain-lac* **by** *simp*
have *card-ac-on-P*: $\forall$ *ac*. *antichain-on P ac* $\longrightarrow$ *card ac* $\leq$ *card ac*
  using *asm1 largest-antichain-on-def less.prems*(*1*) **by** *auto*
then have $\forall$ *ac*. *antichain-on p-plus ac* $\longrightarrow$ *card ac* $\leq$ *card ac*
  using *p-plus-subset antichain-on-def largest-antichain-on-def*
  by (*meson partial-P preorder-class.order-trans*)
then have *largest-antichain-on p-plus ac*
  using *lac-on-PP* **unfolding** *largest-antichain-on-def*
  by (*meson* ‹*largest-antichain-on P ac*› *antichain-on-def*
    largest-antichain-on-def p-plus-subset preorder-class.order-trans)
then have *cv-PP*: $\exists$ *cv*. *chain-cover-on p-plus cv* $\wedge$ *card cv* $=$ *card ac*
 using *less ppl* **by** (*metis 1 card.empty chain-cover-null finite-PP subset-empty*)
then obtain *cvPP* **where** *cvPP-def*: *chain-cover-on p-plus cvPP*
    card cvPP $=$ *card ac*
 using *ac*(*4*) **by** *auto*
obtain *b* **where** *b-def*: *b* $\in$ *?max b* $\notin$ *ac*
  using *asm1 True asm3 less.prems*(*1*) *less.prems*(*2*) *max-e-nIn-lac*
  using ‹*largest-antichain-on P ac*› *ac*(*3*) **by** *blast*
then have $\forall$ *x* $\in$ *ac*. $\neg$ (*b* $\leq$ *x*)
  unfolding *is-maximal-in-set-iff* **using** *partial-P ac-in-P*
  nless-le **by** *auto*
then have *b-not-in-PM*: *b* $\notin$ *p-minus* **using** *p-minus-def* **by** *simp*
have *b* $\in$ *P* **using** *b-def is-maximal-in-set-iff* **by** *blast*
then have *pml*: *card p-minus* $<$ *card P* **using** *b-not-in-PM*
 by (*metis P-PP-PM Un-upper2 card-mono card-subset-eq less.prems*(*2*) *nat-less-le*)
have *p-min-subset*: *p-minus* $\subseteq$ *P* **using** *p-minus-def* **by** *simp*
have *finite-PM*: *finite p-minus* **using** *asm3 p-min-subset finite-def*
  using *less.prems*(*2*) *rev-finite-subset* **by** *blast*
have *partial-PM*: *partial-order-on p-minus* (*relation-of* ($\leq$) *p-minus*)
  by (*simp add*: *partial-order-on-relation-ofI*)

then have *lac-on-PM*: *antichain-on p-minus ac*
  using *2 antichain-lac antichain-on-def* **by** *simp*
then have ∀ *ac. antichain-on p-minus ac* ⟶ *card ac* ≤ *card ac*
  using *card-ac-on-P P-PP-PM antichain-on-def largest-antichain-on-def*
  by (*metis partial-P sup.coboundedI2*)
then have *largest-antichain-on p-minus ac*
  using *lac-on-PM* ‹*largest-antichain-on P ac*› *antichain-on-def*
      largest-antichain-on-def p-min-subset preorder-class.order-trans
  by *meson*
then have *cv-PM*: ∃ *cv. chain-cover-on p-minus cv* ∧ *card cv* = *card ac*
  using *less pml P-PP-PM* ‹*a* ∈ *P*› *a-not-in-PP finite-PM*
  by *blast*
then obtain *cvPM* **where** *cvPM-def*:
        chain-cover-on p-minus cvPM
        card cvPM = *card ac*
  by *auto*
have *lac-minPP*: *ac* = {*x* . *is-minimal-in-set p-plus x*} (**is** *ac* = *?msPP*)
proof
  show *ac* ⊆ {*x* . *is-minimal-in-set p-plus x*}
  proof
    fix *x*
    assume *asm1*: *x* ∈ *ac*
    then have *x-in-PP*: *x* ∈ *p-plus* **using** *1* **by** *auto*
    obtain *y* **where** *y-def*: *y* ∈ *p-plus y* ≤ *x*
      using *1 asm1* **by** *blast*
    then obtain *a* **where** *a-def*: *a* ∈ *ac a* ≤ *y* **using** *p-plus-def* **by** *auto*
    then have *0*: *a* ∈ *p-plus* **using** *1* **by** *auto*
    then have *I*: *a* ≤ *x* **using** *a-def y-def*(*2*) **by** *simp*
    then have *II*: *a* = *x* **using** *asm1 a-def*(*1*) *antichain-lac* **unfolding** *antichain-def* **by** *simp*
    then have *III*: *y* = *x* **using** *y-def*(*2*) *a-def*(*2*) **by** *simp*
    have ∀ *p* ∈ *p-plus.* (*p* ≤ *x*) ⟶ *p* = *x*
    proof
      fix *p*
      assume *asmP*: *p* ∈ *p-plus*
      show *p* ≤ *x* ⟶ *p* = *x*
      proof
        assume *p* ≤ *x*
        then show *p* = *x*
          using *asmP p-plus-def II a-def*(*1*) *antichain-def antichain-lac*
              local.dual-order.antisym local.order.trans mem-Collect-eq
          by (*smt* (*verit*))
      qed
    qed
    then have *is-minimal-in-set p-plus x* **using** *is-minimal-in-set-iff*
      using *partial-PP*
      using *x-in-PP* **by** *auto*
    then show *x* ∈ {*x* . *is-minimal-in-set p-plus x*}
      using *x-in-PP*

```
      using ‹∀ p∈p-plus. p ≤ x ⟶ p = x› local.is-minimal-in-set-iff by force
    qed
  next
    show {x . is-minimal-in-set  p-plus x} ⊆ ac
    proof
      fix x
      assume asm2: x ∈ {x . is-minimal-in-set  p-plus x}
      then have I: ∀  a ∈ p-plus. (a ≤ x) ⟶ a = x
        using is-minimal-in-set-iff
        by (metis dual-order.not-eq-order-implies-strict  mem-Collect-eq)
      have x ∈ p-plus using asm2
        by (simp add: local.is-minimal-in-set-iff)
      then obtain y where y-def: y ∈ ac y ≤ x using p-plus-def by auto
      then have y ∈ p-plus using 1 by auto
      then have y = x using y-def(2) I by simp
      then show x ∈ ac using y-def(1) by simp
    qed
  qed
  then have card-msPP: card ?msPP = card ac by simp
  then have cvPP-elem-in-lac: ∀  m ∈ ?msPP. ∃  c ∈ cvPP. m ∈ c
    using cvPP-def(1) partial-PP asm3 p-plus-subset
        elem-minset-in-chain elem-ac-in-c
      lac-on-PP
    by (simp add: lac-minPP)
  then have cv-for-msPP: ∀  m ∈ ?msPP. ∃  c ∈ cvPP. (∀  a ∈ c. m ≤ a)
    using elem-minset-in-chain partial-PP assms(3)
        cvPP-def(1) e-minset-lesseq-e-chain
    unfolding  chain-cover-on-def[of p-plus cvPP]
    by fastforce
  have lac-elem-in-cvPP: ∀  c ∈ cvPP. ∃  m ∈ ?msPP. m ∈ c
    using cvPP-def card-msPP minset-ac card-ac-cv-eq
    by (metis P-PP-PM finite-Un lac-minPP lac-on-PP less.prems(2))
  then have ∀  c ∈ cvPP. ∃  m ∈ ?msPP. (∀  a ∈ c. m ≤ a)
    using e-minset-lesseq-e-chain chain-cover-on-def cvPP-def(1)
    by (metis mem-Collect-eq)
  then have cvPP-lac-rel: ∀  c ∈ cvPP. ∃  x ∈ ac. (∀  a ∈ c. x ≤ a)
    using lac-minPP by simp
  have lac-maxPM: ac = {x . is-maximal-in-set p-minus x} (is ac = ?msPM)
  proof
    show ac ⊆ ?msPM
    proof
      fix x
      assume asm1: x ∈ ac
      then have x-in-PM: x ∈ p-minus using 2 by auto
      obtain y where y-def: y ∈ p-minus x ≤ y
        using 2 asm1 by blast
      then obtain a where a-def: a ∈ ac y ≤ a using p-minus-def by auto
      then have I: x ≤ a using y-def(2) by simp
      then have II: a = x
```

```
      using asm1 a-def(1) antichain-lac unfolding antichain-def by simp
    then have III: y = x using y-def(2) a-def(2) by simp
    have ∀ p ∈ p-minus. (x ≤ p) ⟶ p = x
    proof
      fix p
      assume asmP: p ∈ p-minus
      show x ≤ p ⟶ p = x
      proof
        assume x ≤ p
        then show p = x
          using p-minus-def II a-def(1) antichain-def antichain-lac asmP
                dual-order.antisym order.trans mem-Collect-eq
          by (smt (verit))
      qed
    qed
    then have is-maximal-in-set p-minus x
      using partial-PM is-maximal-in-set-iff x-in-PM by force
    then show x ∈ {x. is-maximal-in-set p-minus x}
      using x-in-PM by auto
  qed
next
  show ?msPM ⊆ ac
  proof
    fix x
    assume asm2: x ∈ {x . is-maximal-in-set p-minus x}
    then have I: ∀ a ∈ p-minus. (x ≤ a) ⟶ a = x
      unfolding is-maximal-in-set-iff by fastforce
    have x ∈ p-minus using asm2
      by (simp add: local.is-maximal-in-set-iff)
    then obtain y where y-def: y ∈ ac x ≤ y using p-minus-def by auto
    then have y ∈ p-minus using 2 by auto
    then have y = x using y-def(2) I by simp
    then show x ∈ ac using y-def(1) by simp
  qed
qed
then have card-msPM: card ?msPM = card ac by simp
then have cvPM-elem-in-lac: ∀ m ∈ ?msPM. ∃ c ∈ cvPM. m ∈ c
  using cvPM-def(1) partial-PM asm3 p-min-subset elem-maxset-in-chain
        elem-ac-in-c lac-maxPM lac-on-PM
  by presburger
then have cv-for-msPM: ∀ m ∈ ?msPM. ∃ c ∈ cvPM. (∀ a ∈ c. a ≤ m)
  using elem-maxset-in-chain partial-PM assms(3) cvPM-def(1)
        e-chain-lesseq-e-maxset
  unfolding chain-cover-on-def[of p-minus cvPM]
  by (metis mem-Collect-eq)
have lac-elem-in-cvPM: ∀ c ∈ cvPM. ∃ m ∈ ?msPM. m ∈ c
  using cvPM-def card-msPM
    maxset-ac card-ac-cv-eq finite-subset lac-maxPM lac-on-PM less.prems(2)
    p-min-subset partial-PM
```

by *metis*
then have $\forall\ c \in cvPM.\ \exists\ m \in ?msPM.\ (\forall\ a \in c.\ a \leq m)$
using *e-chain-lesseq-e-maxset chain-cover-on-def cvPM-def(1)*
by (*metis mem-Collect-eq*)
then have *cvPM-lac-rel*: $\forall\ c \in cvPM.\ \exists\ x \in ac.\ (\forall\ a \in c.\ a \leq x)$
using *lac-maxPM* by *simp*
obtain *x cp cm* where *x-cp-cm*: $x \in ac$ $cp \in cvPP$ $(\forall\ a \in cp.\ x \leq a)$
$cm \in cvPM$ $(\forall\ a \in cm.\ a \leq x)$
using *cv-for-msPP cv-for-msPM lac-minPP lac-maxPM assms(1)*
unfolding *largest-antichain-on-def antichain-on-def antichain-def*
by (*metis P-PP-PM Sup-empty Un-empty-right all-not-in-conv chain-cover-on-def*

cvPM-def(1) cvPP-def(1) cvPP-lac-rel lac-elem-in-cvPM less.prems(3))

have $\exists\ f.\ \forall\ cp \in cvPP.\ \exists\ x \in ac.\ f\ cp = x \wedge x \in cp$
— defining a function that maps chains in the p plus chain cover to the element in the largest antichain that belongs to the chain.
using *lac-elem-in-cvPP lac-minPP* by *metis*
then obtain *f* where *f-def*: $\forall\ cp \in cvPP.\ \exists\ x \in ac.\ f\ cp = x \wedge x \in cp$ by *blast*
have *lac-image-f*: $f\ `\ cvPP = ac$
proof
show $(f\ `\ cvPP) \subseteq ac$
proof
fix *y*
assume $y \in (f\ `\ cvPP)$
then obtain *x* where $f\ x = y$ $x \in cvPP$ using *f-def* by *blast*
then have $y \in x$ using *f-def* by *blast*
then show $y \in ac$ using *f-def* ‹$f\ x = y$› ‹$x \in cvPP$› by *blast*
qed
next
show $ac \subseteq (f\ `\ cvPP)$
proof
fix *y*
assume *y-in-lac*: $y \in ac$
then obtain *x* where $x \in cvPP$ $y \in x$
using *cvPP-elem-in-lac lac-minPP* by *auto*
then have $f\ x = y$ using *f-def y-in-lac*
by (*metis antichain-def antichain-lac cvPP-lac-rel*)
then show $y \in (f\ `\ cvPP)$ using ‹$x \in cvPP$› by *auto*
qed
qed
have $\forall\ x \in cvPP.\ \forall\ y \in cvPP.\ f\ x = f\ y \longrightarrow x = y$
proof (*rule ccontr*)
assume $\neg\ (\forall x{\in}cvPP.\ \forall y{\in}cvPP.\ f\ x = f\ y \longrightarrow x = y)$
then have $\exists\ x \in cvPP.\ \exists\ y \in cvPP.\ f\ x = f\ y \wedge x \neq y$ by *blast*
then obtain *z x y* where *z-x-y*: $x \in cvPP$ $y \in cvPP$ $x \neq y$ $z = f\ x$ $z = f\ y$
by *blast*
then have *z-in*: $z \in ac$ using *f-def* by *blast*

then have $\forall\ a \in ac.\ (a \in x \lor a \in y) \longrightarrow a = z$
using *ac-to-c partial-P asm3 p-plus-subset cvPP-def(1)*
lac-on-PP z-x-y(1) z-x-y(2)
by (*metis antichain-def antichain-lac cvPP-lac-rel f-def z-x-y(4) z-x-y(5)*)
then have $\forall\ a \in ac.\ a \neq z \longrightarrow a \notin x \land a \notin y$ **by** *blast*
then have $\forall\ a \in (ac - \{z\}).\ a \in \bigcup\ ((cvPP - \{x\}) - \{y\})$
using *cvPP-def(1) 1* **unfolding** *chain-cover-on-def* **by** *blast*
then have *a*: $(ac - \{z\}) \subseteq \bigcup\ ((cvPP - \{x\}) - \{y\})$ (**is** *?lac-z* $\subseteq$ *?cvPP-xy*)
by *blast*
have *b*: *partial-order-on ?cvPP-xy* (*relation-of* ($\leq$) *?cvPP-xy*)
using *partial-PP cvPP-def(1) partial-order-on-def*
dual-order.eq-iff dual-order.eq-iff
dual-order.trans partial-order-on-relation-ofI
dual-order.trans partial-order-on-relation-ofI
by (*smt* (*verit*))
then have *ac-on-cvPP-xy*: *antichain-on ?cvPP-xy ?lac-z*
using *a lac-on-PP antichain-on-def* **unfolding** *antichain-on-def*
by (*metis DiffD1 antichain-def antichain-lac*)
have *c*: $\forall\ a \in ((cvPP - \{x\}) - \{y\}).\ a \subseteq$ *?cvPP-xy* **by** *auto*
have *d*: $\forall\ a \in ((cvPP - \{x\}) - \{y\}).$ *Complete-Partial-Order.chain* ($\leq$) *a*
using *cvPP-def(1)*
unfolding *chain-cover-on-def chain-on-def*
using *z-x-y(2)* **by** *blast*
have *e*: $\forall\ a \in ((cvPP - \{x\}) - \{y\}).$ *chain-on a ?cvPP-xy*
using *b c d chain-on-def*
by (*metis Diff-iff Sup-upper chain-cover-on-def cvPP-def(1)*)
have *f*: *finite ?cvPP-xy* **using** *finite-PP cvPP-def(1)*
unfolding *chain-cover-on-def chain-on-def*
by (*metis* (*no-types, opaque-lifting*) *Diff-eq-empty-iff Diff-subset*
Un-Diff-cancel Union-Un-distrib finite-Un)
have $\bigcup\ ((cvPP - \{x\}) - \{y\}) =$ *?cvPP-xy* **by** *blast*
then have *cv-on*: *chain-cover-on ?cvPP-xy* $((cvPP - \{x\}) - \{y\})$
using *chain-cover-on-def*[*of ?cvPP-xy* $((cvPP - \{x\}) - \{y\})$]
e chain-on-def **by** *argo*
have *card* $((cvPP - \{x\}) - \{y\}) <$ *card cvPP*
using *z-x-y(1) z-x-y(2) finite-PP cvPP-def(1) chain-cover-on-def finite-UnionD*
by (*metis card-Diff2-less*)
then have *card* $((cvPP - \{x\}) - \{y\}) <$ *card* $(ac - \{z\})$
using *cvPP-def(2) finite-PP finite-lac cvPP-def(1) chain-cover-on-def*
finite-UnionD z-x-y(1) z-x-y(2) z-x-y(3) z-in card-Diff2-1-less
by *metis*
then show *False* **using** *antichain-card-leq ac-on-cvPP-xy cv-on f* **by** *fastforce*
qed
then have *inj-f*: *inj-on f cvPP* **using** *inj-on-def* **by** *auto*
then have *bij-f*: *bij-betw f cvPP ac* **using** *lac-image-f bij-betw-def* **by** *blast*
have $\exists\ g.\ \forall\ cm \in cvPM.\ \exists\ x \in ac.\ g\ cm = x \land x \in cm$
using *lac-elem-in-cvPM lac-maxPM* **by** *metis*
then obtain *g* **where** *g-def*: $\forall\ cm \in cvPM.\ \exists\ x \in ac.\ g\ cm = x \land x \in cm$
by *blast*

have *lac-image-g*: g ‘ $cvPM = ac$
proof
  show g ‘ $cvPM \subseteq ac$
  proof
    fix y
    assume $y \in g$ ‘ $cvPM$
    then obtain x **where** x: $g\ x = y\ x \in cvPM$ **using** *g-def* **by** *blast*
    then have $y \in x$ **using** *g-def* **by** *blast*
    then show $y \in ac$ **using** *g-def* x **by** *auto*
  qed
next
  show $ac \subseteq g$ ‘ $cvPM$
  proof
    fix y
    assume *y-in-lac*: $y \in ac$
    then obtain x **where** x: $x \in cvPM\ y \in x$
      using *cvPM-elem-in-lac lac-maxPM* **by** *auto*
    then have $g\ x = y$ **using** *g-def y-in-lac*
      by (*metis antichain-def antichain-lac cvPM-lac-rel*)
    then show $y \in g$ ‘ $cvPM$ **using** x **by** *blast*
  qed
qed
have $\forall\ x \in cvPM.\ \forall\ y \in cvPM.\ g\ x = g\ y \longrightarrow x = y$
proof (*rule ccontr*)
  assume $\neg\ (\forall x{\in}cvPM.\ \forall y{\in}cvPM.\ g\ x = g\ y \longrightarrow x = y)$
  then have $\exists\ x \in cvPM.\ \exists\ y \in cvPM.\ g\ x = g\ y \wedge x \neq y$ **by** *blast*
  then obtain $z\ x\ y$ **where** *z-x-y*: $x \in cvPM\ \ y \in cvPM$
                    $x \neq y\ z = g\ x\ z = g\ y$ **by** *blast*
  then have *z-in*: $z \in ac$ **using** *g-def* **by** *blast*
  then have $\forall\ a \in ac.\ (a \in x \vee a \in y) \longrightarrow a = z$
    using *ac-to-c partial-P asm3 z-x-y*(*1*) *z-x-y*(*2*)
    by (*metis antichain-def antichain-lac cvPM-lac-rel g-def z-x-y*(*4*) *z-x-y*(*5*))
  then have $\forall\ a \in ac.\ a \neq z \longrightarrow a \notin x \wedge a \notin y$ **by** *blast*
  then have $\forall\ a \in (ac - \{z\}).\ a \in \bigcup\ ((cvPM - \{x\}) - \{y\})$
    using *cvPM-def*(*1*) *2* **unfolding** *chain-cover-on-def* **by** *blast*
 then have a: $(ac - \{z\}) \subseteq \bigcup\ ((cvPM - \{x\}) - \{y\})$ (**is** *?lac-z* $\subseteq$ *?cvPM-xy*)

    by *blast*
  have b: *partial-order-on ?cvPM-xy* (*relation-of* $(\leq)$ *?cvPM-xy*)
    using *partial-PP partial-order-on-def*
    by (*smt* (*verit*) *local.dual-order.eq-iff*
        local.dual-order.trans partial-order-on-relation-ofI)
  then have *ac-on-cvPM-xy*: *antichain-on ?cvPM-xy ?lac-z*
    using a *antichain-on-def* **unfolding** *antichain-on-def*
    by (*metis DiffD1 antichain-def antichain-lac*)
  have c: $\forall\ a \in ((cvPM - \{x\}) - \{y\}).\ a \subseteq$ *?cvPM-xy* **by** *auto*
  have d: $\forall\ a \in ((cvPM - \{x\}) - \{y\}).$ *Complete-Partial-Order.chain* $(\leq)$ a
    using *cvPM-def*(*1*)
    unfolding *chain-cover-on-def chain-on-def*

    by (*metis DiffD1*)
  have *e*: $\forall\ a \in ((cvPM - \{x\}) - \{y\}).$ *chain-on a ?cvPM-xy*
    using *b c d chain-on-def*
    by (*metis Diff-iff Union-upper chain-cover-on-def cvPM-def*(*1*))
  have *f*: *finite ?cvPM-xy* **using** *finite-PM cvPM-def*(*1*)
    unfolding *chain-cover-on-def chain-on-def*
    by (*metis* (*no-types*, *opaque-lifting*) *Diff-eq-empty-iff Diff-subset*
        Un-Diff-cancel Union-Un-distrib finite-Un)
  have $\bigcup\ ((cvPM - \{x\}) - \{y\})$ = *?cvPM-xy* **by** *blast*
  then have *cv-on*: *chain-cover-on ?cvPM-xy* $((cvPM - \{x\}) - \{y\})$
    using *chain-cover-on-def e* **by** *simp*
  have *card* $((cvPM - \{x\}) - \{y\}) <$ *card cvPM*
      using *z-x-y*(*1*) *z-x-y*(*2*) *finite-PM cvPM-def*(*1*) *chain-cover-on-def finite-UnionD*
    by (*metis card-Diff2-less*)
  then have *card* $((cvPM - \{x\}) - \{y\}) <$ *card* $(ac - \{z\})$
    using *cvPM-def*(*2*) *finite-PM finite-lac cvPM-def*(*1*) *chain-cover-on-def*
        finite-UnionD z-x-y(*1*) *z-x-y*(*2*) *z-x-y*(*3*) *z-in card-Diff2-1-less*
    by *metis*
  then show *False* **using** *antichain-card-leq ac-on-cvPM-xy cv-on f* **by** *fastforce*
qed
then have *inj-g*: *inj-on g cvPM* **using** *inj-on-def* **by** *auto*
then have *bij-g*: *bij-betw g cvPM ac* **using** *lac-image-g bij-betw-def* **by** *blast*
define *h* **where** *h* = *inv-into cvPP f*
then have *bij-h*: *bij-betw h ac cvPP*
  using *f-def bij-f bij-betw-inv-into* **by** *auto*
define *i* **where** *i* = *inv-into cvPM g*
then have *bij-i*: *bij-betw i ac cvPM*
  using *g-def bij-f bij-g bij-betw-inv-into* **by** *auto*
obtain *j* **where** *j-def*: $\forall\ x \in ac.\ j\ x = (h\ x) \cup (i\ x)$
  using *h-def i-def f-def g-def bij-h bij-i*
  by (*metis sup-apply*)
have $\forall\ x \in ac.\ \forall\ y \in ac.\ j\ x = j\ y \longrightarrow x = y$
proof (*rule ccontr*)
  assume $\neg\ (\forall\ x \in ac.\ \forall\ y \in ac.\ j\ x = j\ y \longrightarrow x = y)$
  then have $\exists\ x \in ac.\ \exists\ y \in ac.\ j\ x = j\ y \wedge x \neq y$ **by** *blast*
  then obtain *z x y* **where** *z-x-y*: $x \in ac\ y \in ac\ z = j\ x\ z = j\ y\ x \neq y$
    by *auto*
  then have *z-x*: $z = (h\ x) \cup (i\ x)$ **using** *j-def* **by** *simp*
  have $z = (h\ y) \cup (i\ y)$ **using** *j-def z-x-y* **by** *simp*
  then have *union-eq*: $(h\ x) \cup (i\ x) = (h\ y) \cup (i\ y)$ **using** *z-x* **by** *simp*
  have *x-hx*: $x \in (h\ x)$ **using** *h-def f-def bij-f bij-h*
    by (*metis bij-betw-apply f-inv-into-f lac-image-f z-x-y*(*1*))
  have *x-ix*: $x \in (i\ x)$ **using** *i-def g-def bij-g bij-i*
    by (*metis bij-betw-apply f-inv-into-f lac-image-g z-x-y*(*1*))
  have $y \in (h\ y)$ **using** *h-def f-def bij-f bij-h*
    by (*metis bij-betw-apply f-inv-into-f lac-image-f z-x-y*(*2*))
  then have $y \in (h\ x) \cup (i\ x)$ **using** *union-eq* **by** *simp*
  then have *y-in*: $y \in (h\ x) \vee y \in (i\ x)$ **by** *simp*

```
    then show False
    proof (cases y ∈ (h x))
      case True
      have ∃ c ∈ cvPP. (h x) = c using h-def f-def bij-h bij-f
        by (simp add: bij-betw-apply z-x-y(1))
      then obtain c where c-def: c ∈ cvPP (h x) = c by simp
      then have x ∈ c ∧ y ∈ c using x-hx True by simp
      then have x = y using z-x-y(1) z-x-y(2) asm1 c-def(1) cvPP-def less.prems
ac
        unfolding largest-antichain-on-def antichain-on-def antichain-def
             chain-cover-on-def chain-on-def chain-def
        by (metis)
      then show ?thesis using z-x-y(5) by simp
    next
      case False
      then have y-ix: y ∈ (i x) using y-in by simp
      have ∃ c ∈ cvPM. (i x) = c using i-def g-def bij-i bij-g
        by (simp add: bij-betw-apply z-x-y(1))
      then obtain c where c-def: c ∈ cvPM (i x) = c by simp
      then have x ∈ c ∧ y ∈ c using x-ix y-ix by simp
      then have x = y
        using z-x-y(1) z-x-y(2) asm1 ac c-def(1) cvPM-def less.prems
        unfolding largest-antichain-on-def antichain-on-def antichain-def
             chain-cover-on-def chain-on-def chain-def
        by (metis)
      then show ?thesis using z-x-y(5) by simp
    qed
  qed
  then have inj-j: inj-on j ac using inj-on-def by auto
  obtain cvf where cvf-def: cvf = {j x | x . x ∈ ac} by simp
  then have cvf = j ` ac by blast
  then have bij-j: bij-betw j ac cvf using inj-j bij-betw-def by auto
  then have card-cvf: card cvf = card ac
    by (metis bij-betw-same-card)
  have j-h-i: ∀ x ∈ ac. ∃ cp ∈ cvPP. ∃ cm ∈ cvPM. (h x = cp) ∧ (i x = cm)
                 ∧ (j x = (cp ∪ cm))
    using j-def bij-h bij-i by (meson bij-betwE)
  have ⋃ cvf = (p-plus ∪ p-minus)
  proof
    show ⋃ cvf ⊆ (p-plus ∪ p-minus)
    proof
      fix y
      assume y ∈ ⋃ cvf
      then obtain z where z-def: z ∈ cvf y ∈ z by blast
      then obtain cp cm where cp-cm: cp ∈ cvPP cm ∈ cvPM z = (cp ∪ cm)
        using cvf-def h-def i-def j-h-i by blast
      then have y ∈ cp ∨ y ∈ cm using z-def(2) by simp
       then show y ∈ (p-plus ∪ p-minus) using cp-cm(1) cp-cm(2) cvPP-def
cvPM-def
```

```
        unfolding chain-cover-on-def chain-on-def by blast
      qed
    next
      show (p-plus ∪ p-minus) ⊆ ⋃ cvf
      proof
        fix y
        assume y ∈ (p-plus ∪ p-minus)
        then have y-in: y ∈ p-plus ∨ y ∈ p-minus by simp
        have p-plus = ⋃ cvPP ∧ p-minus = ⋃ cvPM using cvPP-def cvPM-def
          unfolding chain-cover-on-def by simp
        then have y ∈ (⋃ cvPP) ∨ y ∈ (⋃ cvPM) using y-in by simp
        then have ∃ cp ∈ cvPP. ∃ cm ∈ cvPM. (y ∈ cp) ∨ (y ∈ cm)
          using cvPP-def cvPM-def
          by (meson Union-iff x-cp-cm(2) x-cp-cm(4))
        then obtain cp cm where cp-cm: cp ∈ cvPP cm ∈ cvPM y ∈ (cp ∪ cm)
by blast
        have 1: ∃ cm ∈ cvPM. ∃ x ∈ ac. (x ∈ cp) ∧ (x ∈ cm)
          using cp-cm(1) f-def cvPM-elem-in-lac lac-maxPM by metis
        have 2: ∃ cp ∈ cvPP. ∃ x ∈ ac. (x ∈ cp) ∧ (x ∈ cm)
          using cp-cm(2) g-def cvPP-elem-in-lac lac-minPP
          by meson
        then show y ∈ ⋃ cvf
        proof (cases y ∈ cp)
          case True
          obtain x cmc where x-cm: x ∈ ac x ∈ cp x ∈ cmc cmc ∈ cvPM
            using 1 by blast
          have f cp = x using cp-cm(1) x-cm(1) f-def
            by (metis antichain-def antichain-lac cvPP-lac-rel x-cm(2))
          then have h-x: h x = cp using h-def cp-cm(1) inj-f by auto
          have g cmc = x using x-cm(4) x-cm(1) g-def
            by (metis antichain-def antichain-lac cvPM-lac-rel x-cm(3))
          then have i-x: i x = cmc using i-def
            by (meson bij-betw-inv-into-left bij-g x-cm(4))
          then have j x = h x ∪ i x using j-def x-cm(1) by simp
          then have (h x ∪ i x) ∈ cvf using cvf-def x-cm(1) by auto
          then have (cp ∪ cmc) ∈ cvf using h-x i-x by simp
          then show ?thesis using True by blast
        next
          case False
          then have y-in: y ∈ cm using cp-cm(3) by simp
          obtain x cpc where x-cp: x ∈ ac x ∈ cm x ∈ cpc cpc ∈ cvPP
            using 2 by blast
          have g cm = x using cp-cm(2) x-cp(1) x-cp(2) g-def
            by (metis antichain-def antichain-lac cvPM-lac-rel)
          then have x-i: i x = cm using i-def x-cp(1)
            by (meson bij-betw-inv-into-left bij-g cp-cm(2))
          have f cpc = x using x-cp(4) x-cp(1) x-cp(3) f-def
            by (metis antichain-def antichain-lac cvPP-lac-rel)
          then have x-h: h x = cpc using h-def x-cp(1) inj-f x-cp(4) by force
```

then have *j x* = *h x* ∪ *i x* **using** *j-def x-cp*(*1*) **by** *simp*
then have (*h x* ∪ *i x*) ∈ *cvf* **using** *cvf-def x-cp*(*1*) **by** *auto*
then have (*cpc* ∪ *cm*) ∈ *cvf* **using** *x-h x-i* **by** *simp*
then show *?thesis* **using** *y-in* **by** *blast*
qed
qed
qed
then have *cvf-P*: ⋃ *cvf* = *P* **using** *P-PP-PM* **by** *simp*
have ∀ *x* ∈ *cvf*. *chain-on x P*
proof
fix *x*
assume *asm1*: *x* ∈ *cvf*
then obtain *a* **where** *a-def*: *a* ∈ *ac j a* = *x* **using** *cvf-def* **by** *blast*
then obtain *cp cm* **where** *cp-cm*: *cp* ∈ *cvPP cm* ∈ *cvPM h a* = *cp* ∧ *i a* = *cm*
using *h-def i-def bij-h bij-i j-h-i* **by** *blast*
then have *x-union*: *x* = (*cp* ∪ *cm*) **using** *j-def a-def* **by** *simp*
then have *a-in*: *a* ∈ *cp* ∧ *a* ∈ *cm* **using** *cp-cm h-def f-def i-def g-def*
by (*metis* ‹*a* ∈ *ac*› *bij-betw-inv-into-right bij-f bij-g*)
then have *a-rel-cp*: ∀ *b* ∈ *cp*. (*a* ≤ *b*)
using *a-def*(*1*) *cp-cm*(*1*) *lac-minPP e-minset-lesseq-e-chain*
by (*metis antichain-def antichain-lac cvPP-lac-rel*)
have *a-rel-cm*: ∀ *b* ∈ *cm*. (*b* ≤ *a*)
using *a-def*(*1*) *cp-cm*(*2*) *lac-maxPM e-chain-lesseq-e-maxset a-in*
by (*metis antichain-def antichain-lac cvPM-lac-rel*)
then have ∀ *a* ∈ *cp*. ∀ *b* ∈ *cm*. (*b* ≤ *a*) **using** *a-rel-cp* **by** *fastforce*
then have ∀ *x* ∈ (*cp* ∪ *cm*). ∀ *y* ∈ (*cp* ∪ *cm*). (*x* ≤ *y*) ∨ (*y* ≤ *x*)
using *cp-cm*(*1*) *cp-cm*(*2*) *cvPP-def cvPM-def*
unfolding *chain-cover-on-def chain-on-def chain-def*
by (*metis Un-iff*)
then have *Complete-Partial-Order.chain* (≤) (*cp* ∪ *cm*) **using** *chain-def* **by** *auto*
then have *chain-x*: *Complete-Partial-Order.chain* (≤) *x* **using** *x-union* **by** *simp*
have *x* ⊆ *P* **using** *cvf-P asm1* **by** *blast*
then show *chain-on x P* **using** *chain-x partial-P chain-on-def* **by** *simp*
qed
then have *chain-cover-on P cvf* **using** *cvf-P chain-cover-on-def*[*of P cvf*] **by** *simp*
then show *caseTrue*: *?thesis* **using** *card-cvf ac* **by** *auto*
next — the largest antichain is equal to the maximal set or the minimal set
case *False*
assume ¬ (∃ *ac*. (*antichain-on P ac* ∧ *ac* ≠ *?min* ∧ *ac* ≠ *?max*) ∧ *card ac* = *card lac*)
then have ¬ ((*lac* ≠ *?max*) ∧ (*lac* ≠ *?min*))
using *less*(*2*) **unfolding** *largest-antichain-on-def*
by *blast*
then have *max-min-asm*: (*lac* = *?max*) ∨ (*lac* = *?min*) **by** *simp*
then have *caseAsm*:

$\forall$ *ac*. (*antichain-on P ac* $\wedge$ *ac* $\neq$ *?min* $\wedge$ *ac* $\neq$ *?max*) $\longrightarrow$ *card ac* $\leq$ *card lac*
  using *asm1 largest-antichain-on-def less.prems*(*1*) **by** *presburger*
then have *case2*: $\forall$ *ac*. (*antichain-on P ac* $\wedge$ *ac* $\neq$ *?min* $\wedge$ *ac* $\neq$ *?max*) $\longrightarrow$ *card ac* $<$ *card lac*
  using *False* **by** *force*
obtain *x* **where** *x*: *x* $\in$ *?min*
  using *is-minimal-in-set-iff non-empty-minset partial-P assms*(*2*,*3*)
  by (*metis empty-Collect-eq less.prems*(*2*) *less.prems*(*3*) *mem-Collect-eq*)
then have *x* $\in$ *P* **using** *is-minimal-in-set-iff* **by** *simp*
then obtain *y* **where** *y*: *y* $\in$ *?max x* $\leq$ *y* **using** *partial-P max-elem-for-P*
  using *less.prems*(*2*) **by** *blast*
define *PD* **where** *PD-def*: *PD* = *P* $-$ $\{x,y\}$
then have *finite-PD*: *finite PD* **using** *asm3 finite-def*
  by (*simp add*: *less.prems*(*2*))
then have *partial-PD*: *partial-order-on PD* (*relation-of* ($\leq$) *PD*)
  using *partial-P partial-order-on-def*
  by (*simp add*: *partial-order-on-relation-ofI*)
then have *max-min-nPD*: $\neg$ (*?max* $\subseteq$ *PD*) $\wedge$ $\neg$ (*?min* $\subseteq$ *PD*)
  using *PD-def x y*(*1*) **by** *blast*
have *a1*: $\forall$ *a* $\in$ *P*. (*a* $\neq$ *x*) $\wedge$ (*a* $\neq$ *y*) $\longrightarrow$ *a* $\in$ *PD*
  using *PD-def* **by** *blast*
  then have $\forall$ *a* $\in$ *?max*. (*a* $\neq$ *x*) $\wedge$ (*a* $\neq$ *y*) $\longrightarrow$ *a* $\in$ *PD*
  using *is-maximal-in-set-iff* **by** *blast*
then have (*?max* $-$ $\{x, y\}$) $\subseteq$ *PD* (**is** *?maxPD* $\subseteq$ *PD*) **by** *blast*
have *card-maxPD*: *card* (*?max* $-$ $\{x,y\}$) = (*card ?max* $-$ *1*) **using** *x y*
  proof *cases*
    assume *x* = *y*
    then show *?thesis* **using** *y*(*1*) **by** *force*
  next
    assume $\neg$ (*x* = *y*)
    then have *x* $<$ *y* **using** *y*(*2*) **by** *simp*
    then have $\neg$ (*is-maximal-in-set P x*) **using** *x y*(*1*)
      using ‹*x* $\neq$ *y*› *is-maximal-in-set-iff* **by** *fastforce*
    then have *x* $\notin$ *?max* **by** *simp*
    then show *?thesis* **using** *y*(*1*) **by** *auto*
  qed
  have $\forall$ *a* $\in$ *?min*. (*a* $\neq$ *x*) $\wedge$ (*a* $\neq$ *y*) $\longrightarrow$ *a* $\in$ *PD*
    using *is-minimal-in-set-iff a1*
    by (*simp add*: *a1 local.is-minimal-in-set-iff*)
then have (*?min* $-$ $\{x, y\}$) $\subseteq$ *PD* (**is** *?minPD* $\subseteq$ *PD*) **by** *blast*
have *card-minPD*: *card* (*?min* $-$ $\{x,y\}$) = (*card ?min* $-$ *1*) **using** *x y*
proof *cases*
  assume *x* = *y*
  then show *?thesis* **using** *x* **by** *auto*
next
  assume $\neg$ (*x* = *y*)
  then have *x* $<$ *y* **using** *y*(*2*) **by** *simp*
  then have $\neg$ (*is-minimal-in-set P y*) **using** *is-minimal-in-set-iff x y*(*1*)
    by *force*

```
    then have y ∉ ?min  by simp
    then show ?thesis using x
      by (metis Diff-insert Diff-insert0 card-Diff-singleton-if)
  qed
  then show ?thesis
  proof cases
    assume asm:lac = ?max — case where the largest antichain is the maximal
set
    then have card-maxPD: card ?maxPD = (card lac − 1) using card-maxPD
by auto
    then have ac-less: ∀ ac. (antichain-on P  ac ∧ ac ≠ ?max ∧ ac ≠ ?min)
                  ⟶ card ac ≤ (card lac − 1)
      using case2 by auto
    have PD-sub: PD ⊂ P using PD-def
      by (simp add: ‹x ∈ P› subset-Diff-insert subset-not-subset-eq)
    then have PD-less: card PD < card P using asm3 card-def
      by (simp add: less.prems(2) psubset-card-mono)
     have maxPD-sub: ?maxPD ⊆ PD
      using PD-def ‹{x. is-maximal-in-set P x} − {x, y} ⊆ PD› by blast
    have ?maxPD ⊆ ?max by blast
   then have antichain  ?maxPD using maxset-ac unfolding antichain-def by
blast
    then have ac-maxPD:  antichain-on PD  ?maxPD
      using maxPD-sub antichain-on-def partial-PD by simp
    have acPD-nMax-nMin: ∀ ac . (antichain-on PD ac) ⟶ (ac ≠ ?max ∧ ac
≠ ?min)
      using max-min-nPD antichain-on-def
      by auto
    have ∀ ac. (antichain-on PD  ac) ⟶ (antichain-on P  ac)
      using antichain-on-def antichain-def
      by (meson PD-sub partial-P psubset-imp-subset subset-trans)
    then have ∀ ac. (antichain-on PD  ac) ⟶ card ac ≤ (card lac − 1)
      using ac-less PD-sub max-min-nPD acPD-nMax-nMin by blast
    then have maxPD-lac: largest-antichain-on PD ?maxPD
      using largest-antichain-on-def ac-maxPD card-maxPD by simp
    then have ∃ cv. chain-cover-on PD  cv ∧ card cv = card ?maxPD
    proof cases
      assume PD ≠ {}
      then show ?thesis using less PD-less maxPD-lac finite-PD by blast
    next
      assume ¬ (PD ≠ {})
      then have PD-empty: PD = {} by simp
      then have ?maxPD = {} using maxPD-sub by auto
      then show ?thesis
        using maxPD-lac PD-empty largest-antichain-card-eq-empty by simp
    qed
    then obtain cvPD where cvPD-def: chain-cover-on PD cvPD
                              card cvPD = card ?maxPD by blast
    then have ⋃ cvPD = PD unfolding chain-cover-on-def by simp
```

then have *union-cvPD*: $\bigcup$ (*cvPD* $\cup$ {{*x*,*y*}}) = *P* **using** *PD-def*
using ‹*x* $\in$ *P*› *y*(*1*) *is-maximal-in-set-iff* **by** *force*
have *chains-cvPD*: $\forall$ *x* $\in$ *cvPD*. *chain-on x P*
using *chain-on-def cvPD-def*(*1*) *PD-sub* **unfolding** *chain-cover-on-def*
by (*meson subset-not-subset-eq subset-trans*)
have {*x*,*y*} $\subseteq$ *P* **using** *x y*
using *union-cvPD* **by** *blast*
then have *xy-chain-on*: *chain-on* {*x*,*y*} *P*
using *partial-P y*(*2*) *chain-on-def chain-def*
by *fast*
define *cvf* **where** *cvf-def*: *cvf* = *cvPD* $\cup$ {{*x*,*y*}}
have *cv-cvf*: *chain-cover-on P cvf*
using *chains-cvPD union-cvPD xy-chain-on* **unfolding** *chain-cover-on-def cvf-def*
by *simp*
have $\neg$ ({*x*,*y*} $\subseteq$ *PD*) **using** *PD-def* **by** *simp*
then have {*x*,*y*} $\notin$ *cvPD* **using** *cvPD-def*(*1*)
unfolding *chain-cover-on-def chain-on-def* **by** *auto*
then have *card* (*cvPD* $\cup$ {{*x*,*y*}}) = (*card ?maxPD*) + *1* **using** *cvPD-def*(*2*) *card-def*
by (*simp add*: ‹$\bigcup$ *cvPD* = *PD*› *finite-PD finite-UnionD*)
then have *card cvf* = (*card ?maxPD*) + *1* **using** *cvf-def* **by** *auto*
then have *card cvf* = *card lac* **using** *card-maxPD asm*
by (*metis Diff-infinite-finite Suc-eq-plus1* ‹{*x*, *y*} $\subseteq$ *P*› *card-Diff-singleton card-Suc-Diff1 finite-PD finite-subset less.prems*(*2*) *maxPD-sub y*(*1*))
then show *?thesis* **using** *cv-cvf* **by** *blast*
next
assume $\neg$ (*lac* = *?max*)
— complementary case where the largest antichain is the minimal set
then have *lac* = *?min* **using** *max-min-asm* **by** *simp*
then have *card-minPD*: *card ?minPD* = (*card lac* $-$ *1*) **using** *card-minPD* **by** *simp*
then have *ac-less*: $\forall$ *ac*. (*antichain-on P ac* $\wedge$ *ac* $\neq$ *?max* $\wedge$ *ac* $\neq$ *?min*)
$\longrightarrow$ *card ac* $\leq$ (*card lac* $-$ *1*)
using *case2* **by** *auto*
have *PD-sub*: *PD* $\subseteq$ *P* **using** *PD-def* **by** *simp*
then have *PD-less*: *card PD* < *card P* **using** *asm3*
using *less.prems*(*2*) *max-min-nPD is-minimal-in-set-iff psubset-card-mono*
by (*metis DiffE PD-def* ‹*x* $\in$ *P*› *insertCI psubsetI*)
have *minPD-sub*: *?minPD* $\subseteq$ *PD* **using** *PD-def* **unfolding** *is-minimal-in-set-iff* **by** *blast*
have *?minPD* $\subseteq$ *?min* **by** *blast*
then have *antichain ?minPD* **using** *minset-ac is-minimal-in-set-iff*
unfolding *antichain-def*
by (*metis DiffD1*)
then have *ac-minPD*: *antichain-on PD ?minPD*
using *minPD-sub antichain-on-def partial-PD* **by** *simp*
have *acPD-nMax-nMin*: $\forall$ *ac* . (*antichain-on PD ac*) $\longrightarrow$ (*ac* $\neq$ *?max* $\wedge$ *ac* $\neq$ *?min*)

using *max-min-nPD antichain-on-def*
by *metis*
have $\forall$ *ac.* (*antichain-on PD ac*) $\longrightarrow$ (*antichain-on P ac*)
using *antichain-on-def antichain-def*
by (*meson PD-sub partial-P subset-trans*)
then have $\forall$ *ac.* (*antichain-on PD ac*) $\longrightarrow$ *card ac* $\leq$ (*card lac* $-$ *1*)
using *ac-less PD-sub max-min-nPD acPD-nMax-nMin* **by** *blast*
then have *minPD-lac*: *largest-antichain-on PD ?minPD*
using *largest-antichain-on-def ac-minPD card-minPD* **by** *simp*
then have $\exists$ *cv. chain-cover-on PD cv* $\wedge$ *card cv* = *card ?minPD*
proof *cases*
assume *PD* $\neq$ {}
then show *?thesis* **using** *less PD-less minPD-lac finite-PD* **by** *blast*
next
assume $\neg$ (*PD* $\neq$ {})
then have *PD-empty*: *PD* = {} **by** *simp*
then have *?minPD* = {} **using** *minPD-sub* **by** *auto*
then show *?thesis*
using *minPD-lac PD-empty largest-antichain-card-eq-empty* **by** *simp*
qed
then obtain *cvPD* **where** *cvPD-def*: *chain-cover-on PD cvPD*
card cvPD = *card ?minPD* **by** *blast*
then have $\bigcup$ *cvPD* = *PD* **unfolding** *chain-cover-on-def* **by** *simp*
then have *union-cvPD*: $\bigcup$ (*cvPD* $\cup$ {{*x*,*y*}}) = *P* **using** *PD-def*
using ‹*x* $\in$ *P*› *y*(*1*)
using *is-maximal-in-set-iff* **by** *force*
have *chains-cvPD*: $\forall$ *x* $\in$ *cvPD. chain-on x P*
using *chain-on-def cvPD-def*(*1*) *PD-sub* **unfolding** *chain-cover-on-def*
by (*meson Sup-le-iff partial-P*)
have {*x*,*y*} $\subseteq$ *P* **using** *x y* **using** *union-cvPD* **by** *blast*
then have *xy-chain-on*: *chain-on* {*x*,*y*} *P*
using *partial-P y*(*2*) *chain-on-def chain-def* **by** *fast*
define *cvf* **where** *cvf-def*: *cvf* = *cvPD* $\cup$ {{*x*,*y*}}
then have *cv-cvf*: *chain-cover-on P cvf*
using *chains-cvPD union-cvPD xy-chain-on* **unfolding** *chain-cover-on-def*
by *simp*
have $\neg$ ({*x*,*y*} $\subseteq$ *PD*) **using** *PD-def* **by** *simp*
then have {*x*,*y*} $\notin$ *cvPD* **using** *cvPD-def*(*1*)
unfolding *chain-cover-on-def chain-on-def* **by** *auto*
then have *card* (*cvPD* $\cup$ {{*x*,*y*}}) = (*card ?minPD*) + *1* **using** *cvPD-def*(*2*)
card-def
by (*simp add*: ‹$\bigcup$ *cvPD* = *PD*› *finite-PD finite-UnionD*)
then have *card cvf* = (*card ?minPD*) + *1* **using** *cvf-def* **by** *auto*
then have *card cvf* = *card lac* **using** *card-minPD*
by (*metis Diff-infinite-finite Suc-eq-plus1*
‹*lac* = {*x. is-minimal-in-set P x*}› ‹{*x*, *y*} $\subseteq$ *P*›
card-Diff-singleton card-Suc-Diff1 finite-PD finite-subset
less.prems(*2*) *minPD-sub x*)
then show *?thesis* **using** *cv-cvf* **by** *blast*

```
    qed
  qed
qed
```

5 Dilworth's Theorem for Chain Covers: Statement and Proof

We show that in any partially ordered set, the cardinality of a largest antichain is equal to the cardinality of a smallest chain cover.

```
theorem Dilworth:
  assumes largest-antichain-on P lac
      and finite P
  shows ∃ cv. (smallest-chain-cover-on P cv) ∧ (card cv = card lac)
proof−
  show ?thesis
   using antichain-card-leq largest-antichard-card-eq assms largest-antichain-on-def
    by (smt (verit, ccfv-SIG) card.empty chain-cover-null le-antisym le-zero-eq
            smallest-chain-cover-on-def)
qed
```

6 Dilworth's Decomposition Theorem

6.1 Preliminaries

Now we will strengthen the result above to prove that the cardinality of a smallest chain decomposition is equal to the cardinality of a largest antichain. In order to prove that, we construct a preliminary result which states that cardinality of smallest chain decomposition is equal to the cardinality of smallest chain cover.

We begin by constructing the function make_disjoint which takes a list of sets and returns a list of sets which are mutually disjoint, and leaves the union of the sets in the list invariant. This function when acting on a chain cover returns a chain decomposition.

```
fun make-disjoint::- set list ⇒ -
  where
 make-disjoint [] = ([])
|make-disjoint (s#ls) = (s − (⋃ (set ls)))#(make-disjoint ls)


lemma finite-dist-card-list:
  assumes finite S
  shows ∃ls. set ls = S ∧ length ls = card S ∧ distinct ls
  using assms  distinct-card finite-distinct-list
  by metis
```

lemma *len-make-disjoint*:*length xs* = *length* (*make-disjoint xs*)
  by (*induction xs*, *simp*+)

We use the predicate *list-all2* (⊆), which checks if two lists (of sets) have equal length, and if each element in the first list is a subset of the corresponding element in the second list.

lemma *subset-make-disjoint*: *list-all2* (⊆) (*make-disjoint xs*) *xs*
  by (*induction xs*, *simp*, *auto*)

lemma *subslist-union*:
 assumes *list-all2* (⊆) *xs ys*
 shows $\bigcup$ (*set xs*) ⊆ $\bigcup$ (*set ys*)
  using *assms* **by**(*induction*, *simp*, *auto*)

lemma *make-disjoint-union*:$\bigcup$ (*set xs*) = $\bigcup$ (*set* (*make-disjoint xs*))
proof
  show $\bigcup$ (*set xs*) ⊆ $\bigcup$ (*set* (*make-disjoint xs*))
    by (*induction xs*, *auto*)
next
  show $\bigcup$ (*set* (*make-disjoint xs*)) ⊆ $\bigcup$ (*set xs*)
    using *subslist-union subset-make-disjoint*
    by (*metis*)
qed

lemma *make-disjoint-empty-int*:
  assumes *X* ∈ *set* (*make-disjoint xs*) *Y* ∈ *set* (*make-disjoint xs*)
 and *X* ≠ *Y*
shows *X* ∩ *Y* = {}
  using *assms*
proof(*induction xs arbitrary*: *X Y*)
  case (*Cons a xs*)
  then show *?case*
  proof(*cases X* ≠ *a* − ($\bigcup$ (*set xs*)) ∧ *Y* ≠ (*a* − ($\bigcup$ (*set xs*))))
    case *True*
    then show *?thesis* **using** *Cons*(*1*)[*of X Y*] *Cons*(*2*,*3*)
      by (*smt* (*verit*, *del-insts*) *Cons.prems*(*3*) *Diff-Int-distrib Diff-disjoint*
        Sup-upper make-disjoint.simps(*2*) *make-disjoint-union inf.idem inf-absorb1*

          inf-commute set-ConsD)
  next
    case *False*
    hence *fa*:*X* = *a* − ($\bigcup$ (*set xs*)) ∨ *Y* = *a* − ($\bigcup$ (*set xs*)) **by** *argo*
    then show *?thesis*
    proof(*cases X* = *a* − ($\bigcup$ (*set xs*)))
      case *True*
      hence *Y* ≠ *a* − ($\bigcup$ (*set xs*)) **using** *Cons*(*4*) **by** *argo*
      hence *Y* ∈ *set* (*make-disjoint xs*) **using** *Cons*(*3*) **by** *simp*
      hence *Y* ⊆ $\bigcup$ (*set* (*make-disjoint xs*)) **by** *blast*
      hence *Y* ⊆ $\bigcup$ (*set xs*) **using** *make-disjoint-union* **by** *metis*

```
      hence X ∩ Y = {} using True by blast
      then show ?thesis by blast
    next
      case False
      hence Y:Y = a − (⋃ (set xs)) using Cons(4) fa by argo
      hence X ≠ a − (⋃ (set xs)) using False by argo
      hence X ∈ set (make-disjoint xs) using Cons(2) by simp
      hence X ⊆ ⋃ (set (make-disjoint xs)) by blast
      hence X ⊆ ⋃ (set xs) using make-disjoint-union by metis
      hence X ∩ Y = {} using Y by blast
      then show ?thesis by blast
    qed
  qed
qed (simp)

lemma chain-subslist:
  assumes ∀ i < length xs. Complete-Partial-Order.chain (≤) (xs!i)
    and list-all2 (⊆) ys xs
  shows ∀ i < length ys. Complete-Partial-Order.chain (≤) (ys!i)
  using assms(2,1)
proof(induction)
  case (Cons x xs y ys)
  then have list-all2 (⊆) xs ys  by auto
  then have le: ∀ i<length xs. Complete-Partial-Order.chain (≤) (xs ! i)
    using Cons by fastforce
  then have x ⊆ y using Cons(1) by auto
  then have Complete-Partial-Order.chain (≤) x using Cons
    using chain-subset by fastforce
  then show ?case using le
    by (metis all-nth-imp-all-set insert-iff list.simps(15) nth-mem)
qed(argo)

lemma chain-cover-disjoint:
  assumes chain-cover-on P (set C)
  shows chain-cover-on P (set (make-disjoint C))
proof−
  have ⋃ (set (make-disjoint C)) = P using make-disjoint-union assms(1)
    unfolding chain-cover-on-def by metis
  moreover have ∀ x∈set (make-disjoint C). x ⊆ P
    using subset-make-disjoint assms unfolding chain-cover-on-def
    using calculation by blast
  moreover have ∀ x∈set (make-disjoint C). Complete-Partial-Order.chain (≤) x

    using chain-subslist assms unfolding chain-cover-on-def chain-on-def
    by (metis in-set-conv-nth subset-make-disjoint)
  ultimately show ?thesis  unfolding chain-cover-on-def chain-on-def by auto
qed

lemma make-disjoint-subset-i:
```

assumes $i <$ *length as*
shows (*make-disjoint* (*as*))!$i \subseteq$ (*as*!i)
using *assms*
proof(*induct as arbitrary*: i)
case (*Cons a as*)
then show *?case*
proof(*cases* $i = 0$)
case *False*
have $i - 1 <$ *length as* **using** *Cons*
using *False* **by** *force*
hence (*make-disjoint as*)! $(i - 1) \subseteq$ *as*!$(i - 1)$ **using** *Cons*(*1*)[*of* $i - 1$]
by *argo*
then show *?thesis* **using** *False* **by** *simp*
qed (*simp*)
qed (*simp*)

Following theorem asserts that the corresponding to the smallest chain cover on a finite set, there exists a corresponding chain decomposition of the same cardinality.

lemma *chain-cover-decompsn-eq*:
assumes *finite P*
and *smallest-chain-cover-on P A*
shows $\exists$ *B*. *chain-decomposition P B* $\wedge$ *card B* = *card A*
proof−
obtain *As* **where** *As*:*set As* = *A length As* = *card A distinct As*
using *assms*
by (*metis chain-cover-on-def finite-UnionD finite-dist-card-list smallest-chain-cover-on-def*)
hence *ccdas*:*chain-cover-on P* (*set* (*make-disjoint As*))
using *assms*(*2*) *chain-cover-disjoint*[*of P As*]
unfolding *smallest-chain-cover-on-def* **by** *argo*
hence *1*:*chain-decomposition P* (*set* (*make-disjoint As*))
using *make-disjoint-empty-int*
unfolding *chain-decomposition-def* **by** *meson*
moreover have *2*:*card* (*set* (*make-disjoint As*)) = *card A*
proof(*rule ccontr*)
assume *asm*:¬ *card* (*set* (*make-disjoint As*)) = *card A*
have *length* (*make-disjoint As*) = *card A*
using *len-make-disjoint As*(*2*) **by** *metis*
then show *False*
using *asm assms*(*2*) *card-length ccdas smallest-chain-cover-on-def*
by *metis*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *smallest-cv-cd*:

assumes *smallest-chain-decomposition P cd*
  and *smallest-chain-cover-on P cv*
  shows *card cv* $\leq$ *card cd*
using *assms* **unfolding** *smallest-chain-decomposition-def chain-decomposition-def*
  smallest-chain-cover-on-def **by** *auto*

lemma *smallest-cv-eq-smallest-cd*:
  assumes *finite P*
    and *smallest-chain-decomposition P cd*
    and *smallest-chain-cover-on P cv*
  shows *card cv* = *card cd*
  using *smallest-cv-cd*[*OF assms*(*2*,*3*)] *chain-cover-decompsn-eq*[*OF assms*(*1*,*3*)]
  by (*metis assms*(*2*) *smallest-chain-decomposition-def*)

6.2 Statement and Proof

We extend the Dilworth's theorem to chain decomposition. The following theorem asserts that size of a largest antichain is equal to the size of a smallest chain decomposition.

theorem *Dilworth-Decomposition*:
  assumes *largest-antichain-on P lac*
    and *finite P*
  shows $\exists$ *cd*. (*smallest-chain-decomposition P cd*) $\wedge$ (*card cd* = *card lac*)
  using *Dilworth*[*OF assms*] *smallest-cv-eq-smallest-cd assms*
  by (*metis* (*mono-tags*, *lifting*) *cd-cv chain-cover-decompsn-eq*
    smallest-chain-cover-on-def smallest-chain-decomposition-def)

end

end

Acknowledgement

We would like thank Divakaran D. for valuable suggestions.

References

[1] M. Desharnais. Minimal, maximal, least, and greatest elements w.r.t. restricted ordering. *Archive of Formal Proofs*, October 2024. https://isa-afp.org/entries/Min_Max_Least_Greatest.html, Formal proof development.

[2] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[3] M. A. Perles. A proof of dilworths decomposition theorem for partially ordered sets. *Israel Journal of Mathematics*, 1:105–107, 1963.

[4] A. K. Singh. Fully mechanized proofs of dilworths theorem and mirskys theorem. *arXiv preprint arXiv:1703.06133*, 2017.