Introduction to Greedoids: Formalization using Isabelle/HOL

A Thesis Presented in Partial Fulfillment of
the Honors Master's Degree

Shriya Meenakshisundaram

Abstract

Matroids are the generalization of independence, extended to generic sets. The axioms pertaining to matroids can be applied to algebraic independence in fields, linear independence in vector spaces, set of trees in a graph and other set-theoretical notions of independence. An important generalization of a matroid is the greedoid.

A set system $(E, \mathcal{F})$ consists of an arbitrary nonempty finite set E (known as groundset), and $\mathcal{F}$, a family of subsets of E. A set system is a matroid if $\mathcal{F}$ contains the empty set, contains every subset of each of its elements and satisfies the following: if $X, Y \in \mathcal{F}, |X| > |Y|, \exists x \in X - Y$ such that $Y \cup \{x\} \in \mathcal{F}$. Dropping the second condition gives us the definition of a greedoid.

This project focuses on formalizing definitions and properties of greedoids using the theorem prover Isabelle/HOL. It begins with the definitions of accessible set systems and antimatroids. The second section proceeds to prove theorems relating accessibility, antimatroids and greedoids. The third section talks about an operator τ and its behavior on set systems. It proves an important theorem relating the behavior of τ and accessible set systems.

The last section gives an example of a greedoid taken from graph theory. This example shows that for every digraph, and a fixed vertex r, all the edge sets of directed trees containing r, and the total set of edges form a greedoid. The section then briefly introduces the greedy algorithm of greedoids. Greedy algorithms are algorithms which focus on making locally optimum choices to find a globally optimum solution. The Greedy Algorithm for Greedoids finds an element of $\mathcal{F}$, given a greedoid $(E, \mathcal{F})$ of maximum weight for every modular weight function $c : 2^E \to \mathbb{R}$. It discusses the conditions in which the Greedy Algorithm for Greedoids returns an optimum solution.

Department of Mathematics
Under the supervision of Dr Mohammad Abdulaziz & Dr Lassina Dembele
King's College London
September 2024

Contents

Acknowledgments

I would like to extend my gratitutde to the Department of Mathematics and Department of Informatics at King's College London for giving such an opportunity and approving the same, respectively. I'd also like to convey my heartfelt thanks to Dr Mohammad Abdulaziz and Dr Lassina Dembele, for their constant guidance. Additionally, I thank Dr Jean Lagace, Project Coordinator, Department of Mathematics, for his endearing support and Thomas Ammer, Department of Informatics for his timely help and advice. Lastly, I immensely thank my friends and family, without whom this project would have never neared completion.

1 Set Systems and Accessibility

1.1 Theory: Introduction to Set Systems and Accessibility

Definition 1.1. ([KV06], Definition 2.12.) A set system $(E, \mathcal{F})$ consists of a finite nonempty set E and a set of its subsets $\mathcal{F}$.

Definition 1.2. ([KV06], Section 14.1.) A set system $(E, \mathcal{F})$ is an accessible set system if:
1. $\emptyset \in \mathcal{F}$.
2. $\forall X \in \mathcal{F} - \emptyset$, $\exists x \in X$ such that $X - \{x\} \in \mathcal{F}$.

Definition 1.3. ([KV06], Definition 13.3.) A matroid is a set system $(E, \mathcal{F})$ satisfying the following properties:
1. $\emptyset \in \mathcal{F}$.
2. If $X \subseteq Y \in \mathcal{F}$, then $X \in \mathcal{F}$.
3. If X, $Y \in \mathcal{F}$, $|X| > |Y|$, then $\exists x \in X - Y$ such that $Y \cup \{x\} \in \mathcal{F}$.

Definition 1.4. ([KV06], Section 14.1.) A set system $(E, \mathcal{F})$ is closed under union if for every X, $Y \in \mathcal{F}$, $X \cup Y \in \mathcal{F}$.

Definition 1.5. ([KV06], Section 14.1.) A maximal set X with respect to a predicate P is a set such that there exists no other set Y such that $P(X)$, $P(Y)$ and $X \subset Y$ are true.

Theorem 1.1. ([KV06], Proposition 14.3.) Given an accessible set system $(E, \mathcal{F})$, for $X \in \mathcal{F}$, $|X| = k$, there exists an order $x_1, x_2, \ldots x_k$ for elements of X such that $\forall i \leq k$, $\{x_1, \ldots x_i\} \in \mathcal{F}$.

Proof. We prove this statement by strong induction on the cardinality of an arbitrary X. When $|X| = 0$, $X = \emptyset$, and the statement is vacuously true as $\emptyset \in \mathcal{F}$ for accessible set systems. Let the statement be true for all $k = |X|$. We prove the statement when $|X| = k + 1$. As $X \in \mathcal{F}$, we have $\exists x.\, x \in X$, $X - \{x\} \in \mathcal{F}$ by accessibility definition. We now apply the strong induction hypothesis on $X - \{x\}$ and assign $x_{k+1} = x$ to the order obtained for the set $X - \{x\}$. Hence, the statement is true for $|X| = k + 1$. $\square$

The above theorem is not explicity proven in [KV06] but is mentioned as a statement.

Theorem 1.2. ([KV06], Proposition 14.2.) The following statements are equivalent for an accessible set system $(E, \mathcal{F})$:
1. For all $X \subseteq Y \subset E$ and $z \in E - Y$ such that $X \cup \{z\} \in \mathcal{F}$ and $Y \in \mathcal{F}$, we have $Y \cup \{z\} \in \mathcal{F}$.
2. $\mathcal{F}$ is closed under union.

Proof. $1 \implies 2$: Assume X, $Y \in \mathcal{F}$. The goal is to show that $X \cup Y \in \mathcal{F}$. Let Z be a maximal set such that $X \subseteq Z \subseteq X \cup Y$. Then, the proof of $X \cup Y \in \mathcal{F}$ proceeds by contradiction: assuming $X \cup Y \notin \mathcal{F}$, we have $Y - Z \neq \emptyset$. We now claim that there exists a set $Y' \in \mathcal{F}$, $Y' \subseteq Z$ and an element $y \in Y - Z$ such that $Y' \cup \{y\} \in \mathcal{F}$. Representing set Y as $\{y_1, \ldots y_k\}$ (satisfying Theorem 1.1), where $k = |Y|$ the proof of this claim takes the following cases:
Case 1. $y_1 \in Y - Z$: In this case, consider $\emptyset \in \mathcal{F}$, $\emptyset \subseteq Z$ and $\emptyset \cup \{y_1\} = \{y_1\} \in \mathcal{F}$ (by Theorem 1.1). Hence, $Y' = \emptyset$ proves our claim.
Case 2. $y_1 \notin Y - Z$: Splitting Y as $Y = (Y \cap Z) \cup (Y - Z)$, we have $y_1 \in Y \cap Z$. Now, all other elements $y_2, \ldots, y_k$ either belong to $Y \cap Z$ or $Y - Z$. Now we find a j such that $y_1, \ldots y_j \in Y \cap Z$ and $y_{j+1} \in Y - Z$. We observe that $\{y_1, \ldots y_j\} \in \mathcal{F}$ by Theorem 1.1, $\{y_1, \ldots y_j\} \subseteq Z$ and $y_{j+1} \in Y - Z$ such that $\{y_1, \ldots y_j\} \cup \{y_{j+1}\} = \{y_1, \ldots y_{j+1}\} \in \mathcal{F}$. Our claim is proved by setting $Y' = \{y_1, \ldots y_j\}$ and $y = y_{j+1}$.

From the above claim, we can apply statement 1 to $Y' \subseteq Z$ and $y \in Y - Z \subseteq E - Z$ such that $Y' \cup \{y\} \in \mathcal{F}$. We then obtain $Z \cup \{y\}\ \mathcal{F}$ contradicting the maximality of Z. Hence, $X \cup Y \in \mathcal{F}$.

$2 \implies 1$: Assuming $\mathcal{F}$ to be closed under union, we prove statement 2. Consider sets $X \subseteq Y \subset E$ and an element $z \in E - Y$ such that $X \cup \{z\} \in \mathcal{F}$ and $Y \in \mathcal{F}$, we have $X \cup \{z\} \cup Y = Y \cup \{z\}$ as $X \subseteq Y$. As $\mathcal{F}$ is closed under union, we have $X \cup \{z\} \cup Y = Y \cup \{z\} \in \mathcal{F}$.

□

1.2 Formalization: Introduction to Set Systems and Accessibility

1.2.1 Definitions

The formalization for the above theory begins by defining set systems and accessible set systems.

Listing 1: *Set System and Accessible Definitions*

```
definition "set_system E F = (finite E ∧ (∀ X ∈ F. X ⊆ E))"
definition accessible where "accessible E F ↔ set_system E F ∧ ({} ∈ F) ∧ (∀X. (X ∈ F - {{}}) → (∃x ∈ X. X - {x} ∈ F))"
```

The above definition of `set_system` defines a finite set `E` of `'a` type and a set of subsets of `E` called `F`, followed by the definition of accessibility.

The next set of definitions involve those of maximal sets (Definition 1.5) and set systems closed under union (Definition 1.4).

Listing 2: *Closed Under Union and Maximal Definitions*

```
definition closed_under_union where "closed_under_union F ↔ (∀X Y. X ∈ F ∧ Y ∈ F → X ∪ Y ∈ F)"
definition maximal where "maximal P Z ↔ (P Z ∧ (∄ X. X ⊃ Z ∧ P X))"
```

A usual challenge faced during the formalization of Theorem 1.2 is the conversion of the set theoretical notation in the mathematical proof to Isabelle lists. A few auxiliary lemmas proved in order to ease out this conversion. Now we prove two important theorems relevant to the theory of Theorem 1.2.

1.2.2 Auxiliary Lemmas

The first important proof is that of Theorem 1.1.

Listing 3: *Lemma accessible_property*

```
  lemma accessible_property:
  assumes "accessible E F"
  assumes "X ⊆ E" "X ∈ F"
  shows "∃ l. set l = X ∧ (∀ i. i ≤ length l ⟶ set take i l) ∈ F) ∧ distinct l"
  using assms
proof -
  have "set_system E F" using assms(1) unfolding accessible_def by simp
  then have "finite E" unfolding set_system_def by simp
  then have "finite X" using finite_subset assms(2) by auto
  obtain k where "card X = k" using ⟨finite X⟩ by simp
  then show ?thesis using assms(3)
  proof (induct k arbitrary: X rule: less_induct)
    case (less a)
    then have "card X = a" by simp
    have "X ∈ F" by (simp add: less.prems(2))
    then have "X ⊆ E" using ⟨set_system E F⟩ unfolding set_system_def by simp
    then have "finite X" using ⟨finite E⟩ finite_subset by auto
    then show ?case
```

```
    proof (cases "a = 0")
      case True
      then have "card X = 0" using ⟨card X = a⟩ by simp
      have "¬ (infinite X)" using ⟨finite X⟩ by simp
      then have "X = {}" using ⟨card X = 0⟩ by simp
      then obtain l where l_prop: "set l = X" "distinct l" using
          finite_distinct_list by auto
      then have "l = {}" using l_prop ⟨X = {}⟩ by simp
      have "{} ∈ F" using assms(1) unfolding accessible_def by simp
      then have "∀ i. i ≤ length {} ⟶ set (take i l) ∈ F" using l_prop by
          simp
      then show ?thesis using ⟨l = {}⟩ l_prop by simp
    next
      case False
      then have "X ≠ {}" using ⟨card X = a⟩ by auto
      then have "X ∈ F - {{}}" using ⟨X ∈ F⟩ by simp
      then obtain x where "x ∈ X" "X - {x} ∈ F" using ⟨X ∈ F⟩ assms(1) unfolding
           accessible_def by auto
      have "finite {x}" by simp
      then have factone: "finite (X - {x})" using ⟨finite X⟩ by simp
      have "(X - {x}) ⊂ X" using ⟨x ∈ X⟩ by auto
      then have "card (X - {x}) < card (X)" by (meson ⟨finite X⟩
          psubset_card_mono)
      then have "card (X - {x}) < a" using ⟨card X = a⟩ by simp
      then have "∃ l. set l = X - {x} ∧ (∀ i. i ≤ length l ⟶ set(take i l) ∈
          F) ∧ distinct l" using ⟨X - {x} ∈ F⟩
        using less.hyps by blast
      then obtain l where l_prop: "set l = X - {x} ∧ (∀ i. i ≤ length l ⟶ set
           (take i l) ∈ F) ∧ distinct l" by auto
      let ?l' = l @ [x]
      have conc1: "distinct ?l'" using l_prop by simp
      have l_prop2: "set l = X - {x}" using l_prop by simp
      have "(X - {x}) ∪ {x} = X" using ⟨x ∈ X⟩ by auto
      then have conc2: "(set ?l') = X" using l_prop2 by simp
      have prop2: "(∀ i. i < length ?l' ⟶ set (take i ?l') ∈ F)" using l_prop
          by simp
      have "set (take (length ?l') ?l') ∈ F" using ⟨set ?l' = X⟩ ⟨X ∈ F⟩ by simp
      then have "(∀ i. i ≤ length ?l' ⟶ set (take i ?l') ∈ F)" using prop2
        using antisym_conv2 by blast
       then show ?thesis using conc1 conc2 by fast
     qed
  qed
  qed
```

This property of accessible set systems is proved by strong induction on cardinality of an arbitrary `X ∈ F - {{}}`, following the theory on Theorem 1.1. As `X` is finite, we can perform induction on the cardinality of `X`. The first case is when `card X = 0`, in which case the statement becomes vacuously true as `{} ∈ F`. Now, for `card X ≠ 0`, we obtain a nonempty arbitrary set `X`. We apply accessibility property on `X` to obtain an element `x` such that `x ∈ X` and `X - {x} ∈ F`. Now, we apply the strong induction hypothesis on `X - {x}`. Since its cardinality is lesser than `X`, we can obtain a list `l` such that ∀`i ≤ length l. set (take i l) ∈ F`. We create a new `l @ [x]` and use list comprehension to prove that this is the required list satisfying the accessibility property.

The next theorem proves the existence of a maximal set Z such that X ⊆ Z ⊆ X ∪ Y and Z ∈ F, given X ∈ F and Y ∈ F.

Listing 4: *Lemma exists_maximal*

```
  lemma exists_maximal: assumes "set_system E F" "X ∈ F" "Y ∈ F"
 shows "∃Z. maximal (λ Z. Z ⊇ X ∧ Z ⊆ X ∪ Y ∧ Z ∈ F) Z"
proof -
 let ?S = "{Z. Z ⊇ X ∧ Z ⊆ X ∪ Y ∧ Z ∈ F}"
 have "finite E" using assms(1) unfolding set_system_def by simp
 then have "finite F" using assms(1)
   by (meson Sup_le_iff finite_UnionD rev_finite_subset set_system_def)
 have "?S ⊆ F" by auto
 then have "finite ?S" using ‹finite F› finite_subset by auto
 have "X ∈ F ∧ X ⊆ X ∧ X ⊆ X ∪ Y" using assms(2) by simp
 then have "X ∈ ?S" by simp
 then have "?S ≠ ∅" by auto
 have "∀Z. Z ∈ ?S ⟶ Z ∈ F" by simp
 then have "∀Z. Z ∈ ?S ⟶ Z ⊆ E" using assms(1) unfolding set_system_def by simp
 then have S_prop: "∀Z. Z ∈ ?S ⟶ finite Z" using ‹finite E› finite_subset by (metis (mono_tags, lifting))
 let ?P = "{card Z | Z. Z ⊇ X ∧ Z ⊆ X ∪ Y ∧ Z ∈ F}"
 have "?P ≠ ∅ ∧ finite ?P" using ‹finite ?S› ‹?S ≠ ∅› by simp
 then obtain x where "x = Max ?P" by simp
 then have "x ∈ ?P" using Max_in ‹?P ≠ ∅ ∧ finite ?P› by auto
 then have "∃Z. Z ∈ F ∧ X ⊆ Z ∧ Z ⊆ X ∪ Y ∧ card Z = x" by auto
 then obtain Z where "Z ∈ F ∧ X ⊆ Z ∧ Z ⊆ X ∪ Y ∧ card Z = x" by auto
 have max_prop: "∀z. z ∈ ?P ⟶ z ≤ x" using ‹x = Max ?P› ‹?P ≠ ∅ ∧ finite ?P› by simp
 have "maximal (λ Z. Z ⊇ X ∧ Z ⊆ X ∪ Y ∧ Z ∈ F) Z"
 proof (rule ccontr)
   assume "¬ maximal (λ Z. X ⊆ Z ∧ Z ⊆ X ∪ Y ∧ Z ∈ F) Z"
   then have "∃Z'. Z' ⊃ Z ∧ X ⊆ Z' ∧ Z' ⊆ X ∪ Y ∧ Z' ∈ F" unfolding maximal_def
     using ‹Z ∈ F ∧ X ⊆ Z ∧ Z ⊆ X ∪ Y ∧ card Z = x› by blast
   then obtain Z' where Z'_prop: "Z' ⊃ Z ∧ X ⊆ Z' ∧ Z' ⊆ X ∪ Y ∧ Z' ∈ F" by auto
   then have "Z' ∈ ?S" by simp
   then have "card Z' ∈ ?P" by auto
   have "finite Z'" using S_prop ‹Z' ∈ ?S› by simp
   have "Z ⊂ Z'" using Z'_prop by simp
   then have "card Z < card Z'" using ‹finite Z'› psubset_card_mono by auto
   then show "False" using ‹card Z' ∈ ?P› max_prop
     by (simp add: ‹Z ∈ F ∧ X ⊆ Z ∧ Z ⊆ X ∪ Y ∧ card Z = x› dual_order.strict_iff_not)
 qed
 then show ?thesis by auto
 qed
```

The start of Theorem 1.2 involves stating the existence of this Z. We prove this by taking a set ?S which consists of all sets Z such that Z ∈ F, X ⊆ Z and Z ⊆ X ∪ Y. This set is not empty as it contains X by assumption. It is also finite, as it is a subset of F, a set of subsets of a finite set. We can create another set ?P which contains the cardinalities of all elements of ?S. Since, ?S is finite and nonempty, so is ?P. We can find a set Max ?P ∈ ?P and a set Z ∈ ?S such that card Z = Max ?P. This becomes our required Z. It satisfies all the properties that every element in ?S does. It is also maximal. The proof of this fact is done by contradiction. We assume it is not maximal and obtain another set `Z'` in `?S` that is a strict superset of `Z`. Then the cardinality of this new set `Z'` is greater than that of `Z`, disproving the fact that `card Z = Max ?P`.

1.2.3 Proof of Theorem 1.2

The proof of the second theorem begins by the setting the assumptions of the lemma and proof method.

Listing 5: *Start of proof of lemma second_thm*

```
lemma second_thm:
assumes assum1: "accessible E F"
shows "(∀X Y z. X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z} ∈ F ∧ Y ∈ F → Y ∪ {z} ∈
     F) ↔ closed_under_union F"
proof (intro iffI)
    show "∀X Y z. X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z} ∈ F ∧ Y ∈ F → Y ∪ {z}
       ∈ F
      ⇒ closed_under_union F"
    proof-
      assume assum2: "∀X Y z. X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z} ∈ F ∧ Y ∈ F
          → Y ∪ {z} ∈ F"
      show "closed_under_union F"
        unfolding closed_under_union_def
      proof (rule, rule, rule)
        fix X Y
        assume "X ∈ F ∧ Y ∈ F"
        have "set_system E F" using assum1 unfolding accessible_def by simp
        show  "X ∪ Y ∈ F"
        proof -
```

The overall proof method of the lemma is `intro iffI` which sets two subgoals of the proof as implications in toht directions. For the first direction: if for all $X \subseteq Y \subset E$ and $z \in E - Y$ such that $X \cup \{z\} \in \mathcal{F}$ and $Y \in \mathcal{F}$, we have $Y \cup \{z\} \in \mathcal{F}$, $\mathcal{F}$ is closed under union, the proof method by type `rule, rule, rule` which helps us fix arbitrary variables `X, Y` and prove that `X ∪ Y ∈` $\mathcal{F}$, as shown in the next listing.

Listing 6: *Start of the first implication*

```
show "X ∪ Y ∈ F"
 proof (rule ccontr)
    assume "X ∪ Y ∉ F"
    have "Y - Z ≠ ∅" by (metis ⟨X ∪ Y ∉ F⟩ diff_shunt_var subset_antisym sup.
        bounded_iff z_props)
    have "Y ∈ F" using ⟨X ∈ F ∧ Y ∈ F⟩ by simp
    then have ⟨Y ⊆ E⟩ using ⟨set_system E F⟩ unfolding set_system_def by simp
    have "Z ∈ F" using z_props by simp
    then have ⟨Z ⊆ E⟩ using ⟨set_system E F⟩ unfolding set_system_def by simp
    have "finite E" using ⟨set_system E F⟩ unfolding set_system_def by simp
    then have ⟨finite Z⟩ using ⟨Z ⊆ E⟩ finite_subset by auto
    have ⟨finite Y⟩ using ⟨Y ⊆ E⟩ finite_subset ⟨finite E⟩ by auto
    have "∃ l. set l = Y ∧ (∀i. i ≤ length l → set (take i l) ∈ F) ∧ distinct
        l" using ⟨Y - Z ≠ ∅⟩ ⟨Y ∈ F⟩ accessible_property ⟨Y ⊆ E⟩ assum1 by blast
    then obtain l where l_prop: "set l = Y ∧ (∀i. i ≤ length l → set (take i l
        ) ∈ F) ∧ distinct l" by auto
    then have "set l = Y" by simp
```

```
      have "List.member l (nth l 0)" by (metis Un_absorb2 ‹X ∈ F ∧ Y ∈ F› ‹X ∪ Y ∉ F›‹set l = Y› in_set_member length_pos_if_in_set list_ball_nth subsetI)
      then have "(nth l 0) ∈ Y" using ‹set l = Y› in_set_member by fastforce
      have "Y ≠ ∅" using ‹X ∈ F ∧ Y ∈ F› ‹X ∪ Y ∉ F› by auto
      then have "l ≠ []" using ‹set l = Y› by auto
      then have "length l > 0" by simp
      then have "length l ≥ 1" by linarith
      have Y_split: "Y = (Y - Z) ∪ (Y ∩ Z)" by auto
      then have Y_element_prop: "∀y. y ∈ Y → y ∈ (Y - Z) ∨ y ∈ (Y ∩ Z)" by simp
```

The above listing sets up the proof method, background and facts needed to prove the first implication. This is proved by contradiction as done in Theorem 1.2. We start by assuming `X ∪ Y ∉ F` and establish propterties of `Z`, `Y` and an obtained list `l` where `set l = Y`. The next listing focuses on the proof of the statement: there exists a set $Y' \in \mathcal{F}$, $Y' \subseteq Z$ and an element $y \in Y - Z$ such that $Y' \cup \{y\} \in \mathcal{F}$. This is done by case analysis, as per the informal proof.(Theorem 1.2) The first case`(nth l 0) ∈ Y - Z` is trivial (the empty set satisfies the given conditions) and is skipped from the report. The next listing shows the second case.

Listing 7: *End of first implication*

```
      case False
      then have "(nth l 0) ∈ Y ∩ Z" using ‹l ! 0 ∈ Y› by blast
      then have "Y ∩ Z ≠ {}" by auto
      have "finite (Y ∩ Z)" using ‹finite Y› ‹finite Z› by simp
      then have "∃k. set k = (Y ∪ Z) ∧ k ! 0 = (nth l 0) ∧ distinct k" using exists_list_with_first_element
      ‹(nth l 0) ∈ Y ∩ Z› ‹(nth l 0) ∈ Y ∩ Z› by fast
      then obtain k where k_prop: "set k = Y ∩ Z" "(nth k 0) = (nth l 0) ∧ distinct k" by auto
      then have "k ≠ []" using ‹Y ∩ Z ≠ {} › by auto
      then have first_el_fact: "{nth k 0} = set (take 1 k)" "{nth l 0} = set (take 1 l)" using first_element_set ‹l ≠ []› by auto
      have "distinct l" using l_prop by simp
      have "distinct k" using k_prop(2) by simp
      have "Y ∩ Z ⊂ Y" using ‹Y - Z ≠ {}› by blast
      then have "set k ⊂ set l" using k_prop l_prop by simp
      have "{nth l 0} = {nth k 0}" using k_prop(2) by simp
      then have "set (take 1 l) = set (take 1 k)" using first_el_fact by simp
      then have "∃i. i ≤ length k ∧ set (take i l) ⊆ (set k) ∧ (nth l i) ∈ (set l) - (set k)" using subset_list_el ‹distinct l› ‹distinct k› ‹set k ⊂ set l› by (metis k_prop(2))
      then obtain i where i_prop: "i ≤ length k ∧ set (take i l) ⊆ set (k) ∧ (nth l i) ∈ (set l) - (set k)" by auto
      have "Y - (Y ∩ Z) = Y - Z" by auto
      then have "(set l) - (set k) = Y - Z" using k_prop(1) l_prop by simp
      then have i_prop2: "(nth l i) ∈ Y - Z" using i_prop by simp
      have "card (set k) < card (set l)" using ‹set k ⊂ set l› by (simp add: ‹finite Y› psubset_card_mono)
      then have "length k < length l" using l_prop k_prop(2) by (metis distinct_card)
      then have "i < length l" using i_prop by simp
      then have 1: "set (take i l) ∈ F" using l_prop by simp
      have "i + 1 ≤ length l" using ‹i < length l› by auto
      then have fact_two: "set (take (i+1) l) ∈ F" using l_prop by simp
```

```
    have "set (take i l) ∪ {nth l i} = set (take (i+1) l)" using ⟨i < length l⟩
        set_take_union_nth by simp
    then have 2: "(nth l i) ∈ Y - Z ∧ set (take i l) ∪ {nth l i} ∈ F" using
        fact_two i_prop2 by simp
    have "set (take i l) ⊆ set k" using i_prop by simp
    then have "set (take i l) ⊆ Y ∩ Z" using k_prop by simp
    then have 3: "set (take i l) ⊆ Z" by simp
    then show ?thesis using 1 2 3 by auto
    qed
```

The second case takes the set `Y ∩ Z` and procures a list `k` with distinct elements. Using the lemma `subset_list_el`, we find a minimum `i` such that `set (take l i) ⊆ set k` and `(nth l i) = l`, that is, the first `i` elements lie in `Y ∩ Z` and the `i+1`th element lies in `Y - Z`. This set `set (take i k)` becomes our required `Y'` and we prove the subgoal.

Listing 8: *End of first implication*

```
then obtain Y' where Y'_prop: "Y' ∈ F" "Y' ⊆ Z" "( ∃ y. y ∈ Y - Z ∧ Y' ∪ {y} ∈
     F )" by auto
then obtain y where y_prop: "y ∈ Y - Z" "Y' ∪ {y} ∈ F" by auto
have "Y' ⊆ Z" using Y'_prop by simp
then have "y ∈ E - Z" using y_prop(1) ⟨ Z ⊆ E ⟩ ⟨ Y ⊆ E ⟩ by auto
then have "y ∈ E - Y'" using Y'_prop(2) by auto
then have "Z ∪ {y} ∈ F" using Y'_prop(2) ⟨Z ∈ F⟩ ⟨Z ⊆ E⟩ ⟨ y ∈ E - Z⟩ assum2
    y_prop(2) by blast
have fact_three: "X ⊆ Z ∪ {y}" using z_props by auto
have fact_four: "Z ∪ {y} ⊆ X ∪ Y" using z_props y_prop(1) by simp
have "Z ∪ {y} ⊃ Z" using ⟨y ∈ E - Z⟩ by auto
then show "False" using fact_three ⟨Z ∪ {y} ∈ F ⟩ z_prop fact_four unfolding
    maximal_def by blast
```

The final steps of this implication is done by using the property: for all $X \subseteq Y \subset E$ and $z \in E - Y$ such that $X \cup \{z\} \in \mathcal{F}$ and $Y \in \mathcal{F}$, we have $Y \cup \{z\} \in \mathcal{F}$. Applything the above on `Y', Z, y` from the previous statement, we conclude that we can increase the size of `Z ∈ F`, contradicting the maximality of `Z`.

A concluding remark for the proof of this implication is that the different cases of $y_1 \in Y - Z$ is taken for ease of formalization. While the idea of the proof of the implication is taken from [KV06], the different cases are split according to compatibilty with Isabelle's proof methods.

The reverse implication is proved as follows:

Listing 9: *End of first implication*

```

show 2: "closed_under_union F ⟹ ∀ X Y z. X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z}
     ∈ F ∧ Y ∈ F ⟶ Y ∪ {z} ∈ F"
proof-
  assume "closed_under_union F"
  show "∀ X Y z. X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z} ∈ F ∧ Y ∈ F ⟶ Y ∪ {z}
       ∈ F"
  proof(rule, rule, rule)
    fix X Y z
    show " X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z} ∈ F ∧ Y ∈ F ⟶ Y ∪ {z} ∈ F"
    proof
      assume assum5: "X ⊆ Y ∧ Y ⊆ E ∧ z ∈ E - Y ∧ X ∪ {z} ∈ F ∧ Y ∈ F"
```

```
        then have "X ⊆ Y" by auto
        have "X ∪ {z} ∈ F" using assum5 by auto
        have "Y ∈ F" using assum5 by auto
        have "X ∪ {z} ∪ Y = Y ∪ {z}" using ⟨X ⊆ Y⟩ by auto
        then have "X ∪ {z} ∪ Y ∈ F" using ⟨X ∪ {z} ∈ F⟩ ⟨Y∈F⟩ ⟨closed_under_union
            F⟩ closed_under_union_def by blast
        then show "Y ∪ {z} ∈ F" using ⟨X ∪ {z} ∪ Y = Y ∪ {z}⟩ by auto
        qed
        qed
        qed
```

This subproof is straightforward althrough broken down more in detail compared to the one in [KV06]. We take the step: `X ∪ {z} ∪ Y = Y ∪ {z}` using `X ⊆ Y` and apply the definition of closed under union, to prove the goal.

2 Greedoids and Antimatroids

2.1 Theory: Introduction to Greedoids and Antimatroids

Definition 2.1. ([KV06], Definition 14.1.) A greedoid is a set system $(E, \mathcal{F})$ satisfying:
1. $\emptyset \in \mathcal{F}$.
2. If $X, Y \in \mathcal{F}$, $|X| > |Y|$, then $\exists\, x \in X - Y$ such that $Y \cup \{x\} \in \mathcal{F}$.

Theorem 2.1. ([KV06], Section 14.1) Every greedoid $(E, \mathcal{F})$ is accessible.

Proof. $\emptyset \in \mathcal{F}$ follows from axiom 1 of greedoid. To prove 2, consider $X \in \mathcal{F}$. Let $|X| = k$. Then, set Y $= \emptyset$. By axiom 2 of greedoids, we have an element $x \in X - \emptyset = X$ such that $\emptyset \cup \{x\} = \{x\} \in \mathcal{F}$. Applying axiom 2 once again to X and $\{x\}$ we have an element $y \in X$ such that $\{x, y\} \in \mathcal{F}$. By recursively applying axiom 2 to every new set obtained, we obtain an order for X that satisfies accessible property as in Theorem 1.1. We then take the element x_{k+1} from this order and observe that $X - \{x_{k+1}\} \in \mathcal{F}$, proving the claim. □

The above theorem is not explicitly proven as a theorem in [KV06], but is mentioned as a statement.

Definition 2.2. ([KV06], Section 14.1.) An antimatroid is a set system that satisfies the conditions of Theorem 1.2. In other words, it is a set system that is accessible and closed under union.

Theorem 2.2. ([KV06], Proposition 14.3.) Every antimatroid is a greedoid.

Proof. Antimatroids are accessible and $\emptyset \in \mathcal{F}$ by definition. To prove axiom 2 of greedoids, let $X, Y \in \mathcal{F}$ be such that $|Y| < |X|$. The claim to prove is: $\exists x.\, x \in X - Y$ such that $Y \cup \{x\} \in \mathcal{F}$. We prove this by splitting into the following cases:
Case 1. $X \cap Y = \emptyset$: If this is the case, then $X - Y = X$. Representing elements of $X = \{x_1, \dots x_k\}$ as in Theorem 1.1, where $|X| = k$ we have $x_1 \in X - Y (= X)$, and $\{x_1\} \in \mathcal{F}$ (by Theorem 1.1). As $\mathcal{F}$ is closed under union, $Y \cup \{x_1\} \in \mathcal{F}$. Hence, in this case, x_1 is our required element.
Case 2. $X \cap Y \neq \emptyset$: The proof for this case is split into two subcases:
Subcase (i): $x_1 \in X - Y$: The proof method for this subcase is the same as the one in the first case. We see that $x_1 \in X - Y$, $\{x_1\} \in \mathcal{F}$ (by Theorem 1.1) and hence, $Y \cup \{x_1\} \in \mathcal{F}$ as $\mathcal{F}$ is closed under union. Hence, in this subcase, x_1 is our required element.
Subcase (ii): $x_1 \notin X - Y$: Splitting the set X as $X = (X \cap Y) \cup (X - Y)$, we observe that $x_1 \in (X \cap Y)$. Now, all other elements $x_2, \dots, x_k$ either belong to $X \cap Y$ or $X - Y$. Now we find a j such that $x_1, \dots x_j \in X \cap Y$ and $x_{j+1} \in X - Y$. We observe that $\{x_1, \dots x_j\} \in \mathcal{F}$ by Theorem 1.1 and $\{x_1, \dots x_j\} \subseteq X \cap Y \subseteq Y$. Hence, $Y \cup \{x_{j+1}\} = Y\, \{x_1, \dots x_j\} \cup \{x_{j+1}\} = \{x_1, \dots x_{j+1}\} \in \mathcal{F}$ by Theorem 1.1.

□

2.2 Formalization: Introduction to Greedoids and Antimatroids

The formalization of the above theorem begins with defining antimatroids and setting up the locale for greedoids.

Listing 10: *Greedoids and Antimatroids*

```
locale greedoid =
fixes E :: "'a set"
fixes F :: "'a set set"
assumes contains_empty_set: "{} ∈ F"
assumes third_condition: "(X ∈ F) ∧ (Y ∈ F) ∧ (card X > card Y) ⟹ ∃x ∈ X − Y.
     Y ∪ {x} ∈ F"
assumes ss_assum: "set_system E F"
```

```
assumes acc_assum: "accessible E F"
definition antimatroid where "antimatroid E F ↔ accessible E F ∧ closed_under_union F"
```

The `locale greedoid` fixes a set `E` of type `'a` and a set of sets `F`. It also takes the assumptions of `set_system, acc_assum, third_condition` as per Theorem 2.1 and Definition 2.1.

We now begin the formalization of Theorem 2.2.

Listing 11: *Start of Theorem on Greedoids and Antimatroids*

```
 lemma antimatroid_greedoid:
 assumes assum1: "antimatroid E F"
 shows "greedoid E F"
proof (unfold_locales)
    have 1: "accessible E F ∧ closed_under_union F"
   proof -
     show "accessible E F ∧ closed_under_union F"
       by (meson antimatroid_def assum1)
   qed
   show 2: "set_system E F"
   proof-
     have "accessible E F" using 1 by simp
     then show "set_system E F" unfolding accessible_def by simp
   qed
   show 3: "{} ∈ F" using 1 accessible_def by force
   show 4: "∀ X Y. X ∈ F ∧ Y ∈ F ∧ card Y < card X ⟹ (∃ x ∈ X - Y. Y ∪ {x} ∈ F)"
   proof -
     fix X
     show "∀ Y. X ∈ F ∧ Y ∈ F ∧ card Y < card X ⟹ (∃ x ∈ X - Y. Y ∪ {x} ∈ F)"
     proof -
       fix Y
       assume assum5: "X ∈ F ∧ Y ∈ F ∧ card Y < card X"
       show "(∃ x ∈ X - Y. Y ∪ {x} ∈ F)"
       proof -
```

The lemma assumes set `E` and set of sets `F` to be an antimatroid. The proof method is `unfold_locales`, which means every assumption of the greedoid locale must be proved. We first procure properties `accessible E F` and `closed_under_union E F` from `antimatroid E F`, which helps us prove `{} ∈ F` and `set_system E F`. To prove the assumption `third_condition`, we fix an arbitrary `X` and `Y` and assume `card X` $>$ `card Y`. We then proceed to prove the rest of the theorem on this `X` and `Y`.

Listing 12: *Proof of Theorem on Antimatroids and Greedoids*

```
 show "(∃x ∈ X - Y. Y ∪ {x} ∈ F)"
 proof -
     have "accessible E F" using 1 by auto
     have "closed_under_union F" using 1 by auto
     have "finite E" using 2 unfolding set_system_def by auto
     have "X ∈ F" "Y ∈ F" using assum5 by auto
     have "X ⊆ E" using ⟨X ∈ F⟩ 2 unfolding set_system_def by blast
     have "Y ⊆ E" using ⟨Y ∈ F⟩ ⟨set_system E F⟩ unfolding set_system_def by auto
     then have "finite Y" using ⟨finite E⟩ finite_subset by auto
     have "finite X" using ⟨finite E⟩ ⟨X ⊆ E⟩ finite_subset by auto
```

```
      have "X ≠ {}" using assum5 by auto
      then have "∃ l. set l = X ∧ (∀i. i ≤ length l ⟶ set (take i l) ∈ F) ∧
          distinct l" using ⟨X ∈ F⟩ ⟨accessible E F⟩ accessible_property ⟨X ⊆ E⟩ by
           auto
      then obtain l where l_prop: "set l = X ∧ (∀i. i ≤ length l ⟶ set (take i
          l) ∈ F) ∧ distinct l" by auto
      show "∃x∈X - Y. Y ∪ {x} ∈ F"
```

We begin the proof of third_condition by setting up properties of X, Y and a list l where set l = X. The final statement is proved by case analysis on X ∩ Y = ∅. Case 1 deals with $X \cap Y = \emptyset$.

Listing 13: *Proof of Case 1*

```
  show "∃x∈X - Y. Y ∪ {x} ∈ F"
  proof (cases "X ∩ Y = {}")
  case True
  have "set l = X" using l_prop by auto
  then have "nth l 0 ∈ X" by (metis Int_lower1 True ⟨finite X⟩ assum5 card.empty
      card_length card_seteq gr_zeroI less_nat_zero_code nth_mem)
 then have "nth l 0 ∉ Y" using True by blast
 then have "nth l 0 ∈ X - Y" using ⟨nth l 0 ∈ X⟩ by simp
 have "l ≠ []" using ⟨X ≠ {}⟩ ⟨set l = X⟩ by auto
 then have "length l > 0" by simp
 then have "length l ≥ 1" using linorder_le_less_linear by auto
 then have "set (take 1 l) ∈ F" using l\_prop ⟨set l = X⟩ by simp
 have "{nth l 0} = set (take 1 l)" using first_element_set ⟨l ≠ []⟩ by simp
 have "Y ∪ set (take 1 l) ∈ F" using ⟨set (take 1 l) ∈ F⟩ ⟨Y ∈ F⟩ ⟨
     closed_under_union F⟩
  unfolding closed_under_union_def by blast
 then have "nth l 0 ∈ X - Y ∧ Y ∪ {nth l 0} ∈ F" using ⟨nth l 0 ∈ X - Y⟩ ⟨{nth l
     0} = set (take 1 l)⟩ by auto
then show ?thesis by auto
```

This first case uses the property that the first element of X doesn't lie in Y, satisfying third_condition, following proof of Theorem 2.2. The proof of Case 2, Subcase 1 is the same as Case 1, and is skipped from the report. Now we prove Case 2: Subcase 2. The formalization of this case follows analogously from Theorem 1.2.

Listing 14: *Proof of Case 2 - Subcase 2*

```
  next
    case False
    have "l ≠ []" using ⟨X ≠ {}⟩ ⟨set l = X⟩ by auto
    have "List.member l (nth l 0)" using ⟨set l = X⟩ by (metis ⟨X ≠ {}⟩
        in_set_member length_pos_if_in_set nth_mem subsetI subset_empty)
    then have "nth l 0 ∈ X" using ⟨set l = X⟩ in_set_member by fast
    then have "(nth l 0) ∈ X ∩ Y" using X_element_prop False by simp
    have "finite (X ∩ Y)" using ⟨finite X⟩ ⟨finite Y⟩ by simp
    then have "∃k. set k = X ∩ Y ∧ (nth k 0) = (nth l 0) ∧ distinct k" using
        exists_list_with_first_element ⟨(nth l 0) ∈ X ∩ Y⟩ by fast
    then obtain k where k_prop: "set k = X ∩ Y" "nth l 0 = nth k 0" "distinct k"
        by auto
    then have "k ≠ []" using ⟨X ∩ Y ≠ {}⟩ by auto
    have fact_one: "{nth l 0} = {nth k 0}" using k_prop by simp
    have fact_two: "set (take 1 l) = {nth l 0}" using first_element_set ⟨l ≠ []⟩
        by auto
```

```
    have "set (take 1 k) = {nth k 0}" using first_element_set ⟨k ≠ []⟩ by auto
    then have "set (take 1 l) = set (take 1 k)" using fact_one fact_two by simp
    have "distinct l" using l_prop by simp
    have "distinct k" using k_prop(3) by simp
    have "X ∩ Y ⊂ X" using ⟨X ∩ Y ≠ {}⟩
      by (metis ⟨finite Y⟩ assum5 inf.cobounded1 inf.cobounded2 order.asym psubsetI
         psubset_card_mono)
    then have "(set k) ⊂ (set l)" using l_prop k_prop(1) by simp
    then have assum6: "∃i. i ≤ length k ∧ set (take i l) ⊆ set k ∧ (nth l i) ∈ (
        set l) - (set k)" using subset_list_el
      ⟨distinct l⟩ ⟨distinct k⟩ ⟨(nth l 0) = (nth k 0)⟩ by simp
    then obtain i where i_prop: "i ≤ length k ∧ set (take i k) = set (take i l) ∧
        (nth l i) ∈ (set l) - (set k)" by auto
    have "X - (X ∩ Y) = X - Y" by auto
    then have "(set l) - (set k) = X - Y" using l_prop k_prop(1) by simp
    then have 1: "(nth l i) ∈ X - Y" using i_prop by simp
    then have "(nth l i) ∉ Y" by simp
    have "card (set k) < card (set l)" using ⟨(set k)⊂(set l)⟩ by (simp add: ⟨
        finite X⟩ psubset_card_mono)
    then have "length k < length l" using l_prop k_prop(3)
      by (metis distinct_card)
    then have "i < length l" using i_prop by simp
    then have "set (take i l) ∪ {nth l i} = set (take (i + 1) l)" using
        set_take_union_nth by simp
    have "set (take (i + 1) l) ∈ F" using l_prop ⟨i < length l⟩ by auto
    have "set (take i l) ⊆ set (k)" using i_prop by simp
    then have "set (take i l) ⊆ X ∩ Y" using k_prop by simp
    then have "set (take i l) ⊆ Y" by simp
    then have "Y ∪ {nth l i} = Y ∪ set (take i l) ∪ {nth l i}" using ⟨(nth l i) ∉
        Y⟩ by auto
    also have "... = Y ∪ set (take (i + 1) l)" using ⟨set (take i l) ∪ {nth l i} =
        set (take (i + 1) l)⟩ by auto
    also have "... ∈ F" using ⟨closed_under_union F⟩ ⟨Y ∈ F⟩ ⟨set (take (i + 1) l)
        ∈ F⟩ unfolding closed\_under\_union\_def by simp
    finally have 2: "Y ∪ {nth l i} ∈ F" by simp
    then show ?thesis using 1 2 by auto
  qed
qed
qed
qed
```

In this subcase, we use the fact that `X ∩ Y` is finite and obtain a list `k` such that `set k ⊂ set l`. As the first element of `X`, `(nth l 0)` is in `set k`, we find an `i` such that `set (take i l) ⊆ set k` and `(nth i l) ∈ X - Y`. This becomes our required element and helps in satisfying `third_condition`. A conclusion remark on the formalization of Theorem 2.2 is that it uses the same auxiliary lemmas as the formalization of Theorem 1.2. Additionally, the basic idea of the mathematical proof of Theorem 2.2 is taken from [KV06], but is expanded to the given cases and subcases to be compatible with Isabelle theorem proving.

3 Set System Operators: τ-Operator

This section focuses on one particular set system operator, τ, the properties it satisfies and its behavior on accessible set systems.

3.1 Theory: Set System Operators: τ-Operator

Definition 3.1. ([KV06], Proposition 14.4.) Define operator τ on a set system $(E, \mathcal{F})$ in the following manner:
$\tau(A) = \bigcap\{X \subseteq E \mid A \subseteq X \text{ and } E - X \in \mathcal{F}\}$ for all $A \subseteq E$.

Theorem 3.1. ([KV06], Proposition 14.4., Theorem 13.11.) τ as defined above is a closure operator if and only if it satisfies the following properties:
1. $\forall A \subseteq E$, $A \subseteq \tau(X)$.
2. $\forall A \subseteq B \subseteq E$, $\tau(A) \subseteq \tau(B)$.
3. $\forall A \subseteq E$, $\tau(A) = \tau(\tau(A))$.

Proof. To prove 1, we fix an arbitrary $A \subseteq E$. For all $X \subseteq E$ such that $A \subseteq X$ and $E - X \in \mathcal{F}$, $A \subseteq X$. Hence, $A \subseteq \bigcap\{X \subseteq E \mid A \subseteq X \text{ and } E - X \in \mathcal{F}\}$, thus proving $A \subseteq \tau(A)$.

To prove 2, we fix arbitrary $A \subseteq B \subseteq E$. Consider an arbitrary $X \subseteq E$ such that $B \subseteq X$ and $E - X \in \mathcal{F}$. Then $A \subseteq B \subseteq E$ with $E - X \in \mathcal{F}$. Hence, $\bigcap\{X \subseteq E \mid A \subseteq X \text{ and } E - X \in \mathcal{F}\} \subseteq \bigcap\{X \subseteq E \mid B \subseteq X \text{ and } E - X \in \mathcal{F}\}$, proving $\tau(A) \subseteq \tau(B)$.

Finally, to prove 3, we observe that $\tau(A) \subseteq \tau(\tau(A))$ by 1. We prove the induction in the other direction using the method of contradiction. Suppose $\tau(\tau(A)) \nsubseteq \tau(A)$. We obtain an a such that $a \in \tau(\tau(A)) - \tau(A)$. Now, $a \notin \tau(A)$. Now, $a \in \bigcap\{X \subseteq E \mid A \subseteq X \text{ and } E - X \in \mathcal{F}\}$ and hence $\forall X \subseteq E$ such that $\tau(A) \subseteq X$ and $E - X \in \mathcal{F}$, $a \in X$. Using 1, we have $\forall X \subseteq E$ such that $A \subseteq X$ and $E - X \in \mathcal{F}$, $a \in X$. Hence, $a \in \tau(A)$, a contradiction. □

The next three results are not explicitly stated or proved in [KV06], but are used as facts to prove Theorem 14.5 in [KV06].

Theorem 3.2. ([KV06], Theorem 14.5) For all $A \subseteq E$, $\tau(A) \subseteq E$.

Proof. Recalling the definition of $\tau(A) = \bigcap\{X \subseteq E \mid A \subseteq X \text{ and } E - X \in \mathcal{F}\}$. It is the intersection of subsets X of E satisfying $A \subseteq X$ and $E - X \in \mathcal{F}$. Hence, this intersection also is a subset of E, that is, $\tau(A) \subseteq E$. □

Theorem 3.3. ([KV06], Theorem 14.5) For all $A \subseteq E$, $E - \tau(A) \in \mathcal{F}$, where $\mathcal{F}$ is closed under union.

Proof. By definition, $\tau(A) = \bigcap\{X \subseteq E \mid A \subseteq X \text{ and } E - X \in \mathcal{F}\}$.

For each $X \subseteq E$ such that $A \subseteq x$ and $E - X \in \mathcal{F}$, $E - X$ satisfies $E - X \subseteq E - A$ and $E - X \subseteq \mathcal{F}$. We claim: $E - \tau(A) = \bigcup\{Y \subseteq E \mid A \subseteq E - Y \text{ and } Y \in \mathcal{F}\}$. Let $\{Y \subseteq E \mid A \subseteq E - Y \text{ and } Y \in \mathcal{F}\} = S$. We prove two inclusions for this claim: $E - \tau(A) \subseteq \bigcup S$ and $\bigcup S \subseteq E - \tau(A)$. To prove the first inclusion, $E - \tau(A) \subseteq \bigcup S$, we fix an element $x \in E - \tau(A)$. Since $x \notin \tau(A)$, we obtain an X such that $X \subseteq E$, $A \subseteq X$ and $E - X \in \mathcal{F}$ and $x \notin X$. We conclude that $x \in E - X$, where $E - X \subseteq E$, $A \subseteq E - (E - A)(= A)$ and $E - X \in \mathcal{F}$. Hence, there exists a set in S that contains x that results in $x \in \bigcup S$, thus proving the claim.
To prove the backward inclusion, $\bigcup S \subseteq E - \tau(A)$, fix an element $x \in \bigcup S$. We then obtain a set Y where $Y \subseteq E$, $A \subseteq E - Y$ and $Y \in \mathcal{F}$ such that $x \in Y$ and $x \in E$ by definition. We prove $x \notin \tau(A)$ by contradiction. Assuming $x \in \tau(A)$, we say that $\forall X \subseteq E$ such that $A \subseteq X$ and $E - X \in \mathcal{F}$, $x \in X$. Then we have, $x \in E - Y$ where $E - Y \subseteq E$, $A \subseteq E - Y$ and $E - (E - Y) \in \mathcal{F}$ by the properties of Y. But we have $x \in Y$ (not in $E - Y$) a contradiction. □

Theorem 3.4. ([KV06], Theorem 14.5) For all $A \in \mathcal{F} - \emptyset$, $\tau(E - A) = (E - A)$. Here, $\mathcal{F}$ is closed under union and contains $\emptyset$.

Proof. The forward containment, $(E - A) \subseteq \tau(E - A)$ is direct from axiom 1 of Theorem 3.1. The backward containment is proved by contradiction. Let $\tau(E - A) \nsubseteq (E - A)$. Then we can obtain an element $a \in \tau(E - A) - (E - A)$, that is, $a \in \bigcap\{X \subseteq E \,|\, (E - A) \subseteq X \text{ and } E - X \in \mathcal{F}\}$. Hence, we can say that for every $X \subseteq E$ such that $E - A \subseteq X$ and $E - X \in \mathcal{F}$, $a \in X$. But, $(E - A)$ satisfies the above conditions: $(E - A) \subseteq E$, $(E - A) \subseteq (E - A)$ and $E - (E - A) = A \in \mathcal{F}$. Hence $a \in (E - A)$, a contradiction. □

Theorem 3.5. ([KV06], Theorem 14.5) For a set system $(E, \mathcal{F})$ that is closed under union and $\emptyset \in \mathcal{F}$, $(E, \mathcal{F})$ is accessible if and only if operator τ satisfies the antiexchange property: $\forall X \subseteq E$, $y, z \in E - \tau(X)$, $y \neq z$, and $z \in \tau(X \cup \{y\})$ then $y \notin \tau(X \cup \{z\})$.

Proof. To prove $1 \implies 2$, we assume the given set system is accessible. If $(E, \mathcal{F})$ is accessible, it becomes an antimatroid (by Definition 2.2) as it is closed under union. Hence, $(E, \mathcal{F})$ is a greedoid by Theorem 2.2 and satisfies axiom 2 of greedoids: If $X, Y \in \mathcal{F}$, $|X| > |Y|$, then $\exists x \in X - Y$ such that $Y \cup \{x\} \in \mathcal{F}$. Let $B := E - \tau(X)$ for some $X \subseteq E$ and $y, z \in B$ with $z \in \tau(X \cup \{y\})$. Let $A := E - \tau(X \cup \{y\})$. We observe that $E - \tau(X)\,(= B)$ and $E - \tau(X \cup \{y\})\,(= A) \in \mathcal{F}$ by Theorem 3.3. Also, $\tau(X) \subseteq \tau(X \cup \{y\})$ by Theorem 3.1, and we have $E - \tau(X \cup \{y\}) \subseteq E - \tau(X)$, that is, $A \subseteq B$.
Consider set $B - A$. $B = E - \tau(X) \subseteq E - X$ (as $X \subseteq \tau(X)$, from Theorem 3.1). Hence, $(B - A) \subseteq E - (X \cup A)$. Also $A \subseteq B - \{y, z\}$ as $y, z \notin A$ but in B. Hence, $|A| < |B|$. We proceed to prove $y \notin \tau(X \cup \{z\})$, hence satisfying the antiexchange property.
We now apply axiom 2 of greedoids on A and B. We obtain an element $b \in B - A$ such that $A \cup \{b\} \in \mathcal{F}$.
Observe that $A \cup \{b\} \nsubseteq E - (X \cup \{y\})$. We prove this by contradiction. Assuming $A \cup \{b\} \subseteq E - (X \cup \{y\})$ is true, we subtract both sides from E and get the inequality: $(X \cup \{y\}) \subseteq E - (A \cup \{b\})$. Applying τ operator on both sides, we get $\tau(X \cup \{y\}) \subseteq \tau(E - (A \cup \{b\})) \implies \tau(X \cup \{y\}) \subseteq E - (A \cup \{b\})$. This relation is a contradiction as we know that $E - A = E - (E - \tau(X \cup \{y\})) = \tau(X \cup \{y\})$.
Now, $E - (\tau(X \cup \{y\})) \subseteq E - (X \cup \{y\})$, that is, $A \subseteq E - (X \cup \{y\})$. Using the previously proved relation, we have $A \cup \{b\} \nsubseteq E - (X \cup \{y\})$. Hence, $b \in (X \cup \{y\})$. However, $b \in B - A \subseteq E - (X \cup A)$, as per definition. Hence $b \in E - (X \cup A) \implies b \notin X$. Using this in the relation $b \in (X \cup \{y\})$, we obtain $b = y$. Hence, by the property of b, we have $A \cup \{y\} \in \mathcal{F}$.
Now, we have $z \in (E - A) = \tau(X \cup \{y\})$, $X \subseteq (E - A) = \tau(X \cup \{y\})$, $y \neq z$, and $y \notin X$ (since $y \in E - (\tau(X)) \implies y \notin \tau(X) \implies y \notin X$). Putting together the above facts, we have $(X \cup \{z\}) \subseteq E - (A \cup \{y\})$. Applying τ on both sides gives $\tau(X \cup \{z\}) \subseteq \tau(E - (A \cup \{y\})) \implies \tau(X \cup \{y\}) \subseteq E - (A \cup \{y\})$ as $(A \cup \{y\}) \in \mathcal{F}$ and we can apply Theorem 3.3. This leads to the conclusion that $y \notin \tau(X \cup \{z\})$, proving the antiexchange property.

To prove $2 \implies 1$, we assume $A \in \mathcal{F} - \emptyset$. Let $X = E - A$. We have $\tau(X) = X$ using Theorem 3.3. Let $a \in A$ such that $|\tau(X \cup \{a\})|$ is minimum. Our goal is to show $\tau(X \cup \{a\}) = X \cup \{a\}$. We obtain the forward inclusion from Theorem 3.1, that is, $X \cup \{a\} \subseteq \tau(X \cup \{a\})$. We prove the backward inclusion by contradiction.
Assume $\tau(X \cup \{a\}) \nsubseteq (X \cup \{a\})$. Then we obtain an element b such that $\tau(X \cup \{a\}) - (X \cup \{a\})$, $b \neq a$. We can apply the antiexchange property as $b, a \in A(= E - (E - A) = E - (\tau(X)))$, $b \neq a$ and $b \in \tau(X \cup \{a\})$. We then obtain $a \notin \tau(X \cup \{b\})$.
Now, $X \cup \{b\} \subseteq \tau(X \cup \{a\}) \cup \{b\}$. Applying τ on both sides, we get $\tau(X \cup \{b\}) \subseteq \tau(\tau(X \cup \{a\}) \cup \{b\}) = \tau(\tau(X \cup \{a\})) = (\tau(X \cup \{a\})$ by applying facts $b \in \tau(X \cup \{a\})$ and Theorem 3.1 respectively. Hence we get $(\tau(X \cup \{b\}) \subseteq (\tau(X \cup \{a\})$. Since $a \notin \tau(X \cup \{b\})$ but in $a \in \tau(X \cup \{a\})$, $\tau(X \cup \{b\})$ is a proper subset of $\tau(X \cup \{a\}$, contradicting the choice of a.
Therefore $\tau(X \cup \{a\}) = (X \cup \{a\})$.

Now, we know that $E - \tau(X \cup \{a\}) \in \mathcal{F}$. Hence, $E - (X \cup \{a\}) \in \mathcal{F} \implies E - ((E - A) \cup \{a\}) = A - \{a\}) \in \mathcal{F}$, proving accessiblity property. $\square$

3.2 Formalization: Set System Operators: τ-Operator

The formalization of this section begins with defining the locale for closure operators on set systems, definition of τ, and Theorem 3.1. Then we prove a few auxiliary lemmas, followed by the definition of antiexchange property and Theorem 3.5.

Listing 15: *Set System Operators: Definition of τ and locale*

```
locale closure_operator =
fixes E:: "'a set"
fixes F:: "'a set set"
assumes ss_assum: "set_system E F"
fixes set_system_operator:: "'a set ⇒ 'a set"
assumes S_1: "∀X. X ⊆ E ⟶ X ⊆ set_system_operator X"
assumes S_2: "∀X Y. X ⊆ E ∧ Y ⊆ E ∧ X ⊆ Y ⟶ set_system_operator X ⊆
    set_system_operator Y"
assumes S_3: "∀X. X ⊆ E ⟶ set_system_operator X = set_system_operator (
    set_system_operator X)"

context closure_operator
begin

definition τ :: "'a set ⇒ 'a set" where "τ A = ⋂{X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}"
```

Setting up a locale for closure operators helps us access the fixed sets `E` and `F`, and their properties easily.

3.2.1 Proof of Theorem 3.1

The proof of Theorem 3.1 has the proof method: `unfold_locales`. The proof of the first two parts are fairly straightforward and is formalized in the listing given below:

Listing 16: *Start of τ-Closure Operator proof*

```

lemma τ_closure_operator:
assumes assum1: "closed_under_union F"
assumes assum2: "{} ∈ F"
assumes assum3: "set_system E F"
shows "closure_operator E F τ"

proof (unfold_locales)

show 1: "∀X. X ⊆ E ⟶ X ⊆ τ X"
proof (intro allI impI)
    fix A
    assume "A ⊆ E"
    then have "A ⊆ ⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}" by auto
    then show "A ⊆ τ A"
      unfolding τ_def by blast
  qed
```

```

  show 2: "∀X Y. X ⊆ E ∧ Y ⊆ E ∧ X ⊆ Y ⟶ τ X ⊆ τ Y"
  proof (rule allI)
    fix X'
    show "∀Y. X' ⊆ E ∧ Y ⊆ E ∧ X' ⊆ Y⟶ τ X' ⊆ τ Y"
    proof (rule allI)
      fix Y
      show "X' ⊆ E ∧ Y ⊆ E ∧ X' ⊆ Y ⟶ τ X' ⊆ τ Y"
      proof (rule impI)
        assume assum3: "X' ⊆ E ∧ Y ⊆ E ∧ X' ⊆ Y"
        then have A_B_prop: "⋂ {X. X ⊆ E ∧ X' ⊆ X ∧ E - X ∈ F} ⊆ ⋂ {X. X ⊆ E ∧
             Y ⊆ X ∧ E - X ∈ F}"
          by fastforce
        then show "τ X' ⊆ τ Y" by (simp add: τ_def)
      qed
    qed
   qed
```

The third part takes the proof method `rule ccontr`, and follows the proof by contradiction mentioned in Theorem 3.1.

Listing 17: *End of τ-Closure Operator proof*

```

  show 3: "∀X. X ⊆ E → τ X = τ (τ X)"
  proof (intro allI impI)
      fix X
      assume "X ⊆ E"
      have "τ X ⊆ τ (τ X)" using 1 unfolding τ_def by blast
      have "τ (τ X) ⊆ τ X"
      proof (rule ccontr)
          assume "¬τ (τ X) ⊆ τ X"
          obtain y where assum4: "y ∈ τ (τ X) - τ X" using ‹¬ τ (τ X) ⊆ τ X› by
              auto
          have y_prop: "y ∈ ⋂ {Y. Y ⊆ E ∧ (τ X) ⊆ Y ∧ E - Y ∈ F}" using assum4
              τ_def by auto
          have "y ∉ τ X" using assum4 by auto
          then have Y_prop: "∀Y. Y ⊆ E ∧ τ X ⊆ Y ∧ E - Y ∈ F → y ∈ Y" using
              y_prop by auto
          then have "∀Z. Z ⊆ E ∧ X ⊆ Z ∧ E - Z ∈ F → y ∈ Z" using ‹∀X. X ⊆ E →
               X ⊆ τ X› unfolding τ_def by blast
          then have "y ∈ τ X" unfolding τ_def by blast
          then show "False" using ‹y ∉ τ X› by simp
        qed
          then show "τ X = τ (τ X)" using ‹τ X ⊆ τ (τ X)› by blast
        qed
        show "set_system E F" using assum3 by simp
      qed
```

3.2.2 Auxiliary Lemmas

We now prove Theorem 3.2.

Listing 18: *Set System Operators: Property 1*

```
lemma τ_in_E:
assumes "set_system E F" "A ⊆ E" "{} ∈ F"
shows "τ A ⊆ E"
proof -
    have "A ⊆ E" using assms by simp
    have τ_def_expand: "τ A = ⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}" unfolding τ_def
        by auto
    have "⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F} ⊆ E"
      using assms(2) "{} ∈ F" by fastforce
    then show "τ A ⊆ E" using τ_def_expand by auto
qed
```

By unfolding the definition of τ, we observe how every element of the big intersection lies in E, proving the claim.

Now is the proof of Theorem 3.3 which follows the informal proof given in the previous section.

Listing 19: *Set System Operators: Property 2*

```
  lemma τ_prop:
    assumes "A ⊆ E" "set_system E F" "closed_under_union F" "{} ∈ F"
    shows "E - τ A ∈ F"
  proof -
    have 1: "τ(A) = ⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}" unfolding τ_def by simp
    then have "τ A ⊆ E" using τ_in_E assms(2) assms(4) assms(1) by auto
    let ?S = "{Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F}"
    have 1: "E - τ A = ⋃ ?S"
    proof
      show "E - τ A ⊆ ⋃ {Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F}"
      proof
        fix x
        show "x ∈ E - τ A  ⟹  x ∈ ⋃ {Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F}"
        proof -
          assume "x ∈ E - τ A"
          then have "x ∈ E" and "x ∉ τ A" by auto
          from ‘x ∉ τ A‘ obtain X where "X ⊆ E" "A ⊆ X" "E - X ∈ F" "x ∉ X"
            using ‘τ A = ⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}‘ by auto
          then have "x ∈ E - X" and "E - X ∈ F" using ‘x ∈ E‘ by auto
          then show "x ∈ ⋃ ?S" using ‘X ⊆ E‘ ‘A ⊆ X‘ by auto
        qed
      qed

      show "⋃ {Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F} ⊆ E - τ A"
      proof
        fix x
        show "x ∈ ⋃ {Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F}  ⟹  x ∈ E - τ A"
        proof -
          assume "x ∈ ⋃ ?S"
          then obtain Y where Y_prop: "x ∈ Y" and "Y ⊆ E" and "A ⊆ E - Y" and "Y
              ∈ F" by auto
          then have "x ∈ E" and "x ∉ τ A"
          proof -
            show "x ∈ E" using ‘Y ⊆ E‘ ‘x ∈ Y‘ by auto
            show "x ∉ τ A"
            proof (rule ccontr)
```

```
            assume "¬ (x ∉ τ A)"
            then have "x ∈ τ A" by simp
            then have "x ∈ ⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}" using ‘τ A = ⋂ {X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F}‘by simp
            then have fact_one: "∀X. X ⊆ E ∧ A ⊆ X ∧ E - X ∈ F ⟹ x ∈ X" by simp
            have 1: "E - Y ⊆ E" using ‘Y ⊆ E‘ by simp
            have "E - (E - Y) = Y" using ‘Y ⊆ E‘ by auto
            then have "E - (E - Y) ∈ F" using ‘Y ∈ F‘ by simp
            then have "(E - Y) ⊆ E ∧ A ⊆ (E - Y) ∧ E - (E - Y) ∈ F" using 1 ‘A ⊆ E - Y‘ by simp
            then have "x ∈ E - Y" using fact_one by auto
            then have "x ∉ Y" by simp
            then show False using ‘x ∈Y‘ by auto
          qed
        qed
        then show ?thesis by simp
      qed
    qed
  qed
  have prop1: "{Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F} ⊆ F" by auto
  have "finite E" using assms(2) unfolding set_system_def by simp
  then have "finite F" using assms(2) unfolding set_system_def
    by (meson Sup_le_iff finite_UnionD finite_subset)
  then have "finite {Y. Y ⊆ E ∧ A ⊆ E - Y ∧ Y ∈ F}" using finite_subset by simp
  then have "⋃ ?S ∈ F"
    using closed_under_arbitrary_unions assms(2-4) prop1  by simp
  then show "E - τ A ∈ F" using 1 by simp
qed
```

We now show the last property of τ, which determines the behavior of sets in $\mathcal{F}$ under τ. It follows the proof from Theorem 3.4.

Listing 20: *Set System Operators: Property 2*

```

lemma τ_prop2:
assumes "A ∈ F" "set_system E F" "closed_under_union F" "{} ∈ F"
shows "τ (E - A) = E - A"
proof
  show "E - A ⊆ τ (E - A)"
  proof -
      have "A ⊆ E" using assms unfolding set_system_def by auto
      then have "E - A ⊆ E" by auto
      then show ?thesis using τ_closure_operator assms(3) assms(4) closure_operator.S_1 ss_assum by blast
   qed
   show "τ (E - A) ⊆ E - A"
   proof (rule ccontr)
     assume assum1: "¬(τ (E - A) ⊆ E - A)"
     then obtain x where x_prop: "x ∈ τ (E - A)" "x ∉ E - A" by auto
     have "A ⊆ E" using assms unfolding set_system_def by auto
      then have "E - A ⊆ E" by auto
```

```
      have "x ∈ ⋂ {X. X ⊆ E ∧ (E - A) ⊆ X ∧ E - X ∈ F}" using x_prop(1)
          unfolding τ_def by simp
      then have 1: "∀ X. X ⊆ E ∧ (E - A) ⊆ X ∧ E - X ∈ F  ⟹  x ∈ X" by simp
      have "E - (E - A) = A" using ⟨A ⊆ E⟩ by auto
      then have "E - (E - A) ∈ F" using assms(1) by simp
      then have "E - A ⊆ E ∧ (E - A) ⊆ E - A ∧ (E - (E - A)) ∈ F" using ⟨E - A ⊆
          E⟩ by simp
      then have "x ∈ (E - A)" using 1 by blast
      then show "False" using x_prop(2) by simp
    qed
qed
```

3.2.3 Proof of Theorem 3.5

The proof of Theorem 3.5 starts with the definition of antiexchange property and the proof method for the lemma: `intro iffI`. This sets the two subgoals of the proof as both implications in the if-and-only-if statement.

The forward direction, that is, an accessible set system implies τ satisfies anti-exchange property can be broadly categorized into four parts. The first part sets up the background to prove the antiexchange property. It fixes a subset `X` of `E` and two elements `y` and `z` that lie in `E - τ X`. We then define variables `?A = E - τ(X ∪ {y})` and `?B = E - τ(X)`. On applying accessiblity, we obtain `antimatroid E F`, and hence `greedoid E F`. This helps us apply axiom 2 of greedoids as in Definition 2.1.

Listing 21: *Start of Proof of Accessibility ⇔ Antiexchange Property*

```
 show "accessible E F  ⟹  antiexchange_property τ"
proof -
  assume assum3: "accessible E F"
  have "antimatroid E F" using assum3 assum1 by (simp add: antimatroid_def)
  then have "closed_under_union F" unfolding antimatroid_def by auto
  have "set_system E F" using assum3 unfolding accessible_def by auto
  have "greedoid E F" using ⟨antimatroid E F⟩ antimatroid_greedoid by auto
  then have third_condition: "(X ∈ F) ∧ (Y ∈ F) ∧ (card X > card Y)  ⟹  ∃ x ∈ X
      - Y. Y ∪ {x} ∈ F"
    using greedoid.third_condition by blast
  have contains_empty_set: "{} ∈ F" using assum3 unfolding accessible_def by simp
  show "antiexchange_property τ"
    unfolding antiexchange_property_def
  proof (rule allI)
    fix X
    show "∀ y z. X ⊆ E ∧ y ∈ E - τ X ∧ z ∈ E - τ X ∧ y ≠ z ∧ z ∈ τ (X ∪ {y})
        ⟹  y ∉ τ (X ∪ {z})"
    proof (rule allI)
      fix y
      show "∀ z. X ⊆ E ∧ y ∈ E - τ X ∧ z ∈ E - τ X ∧ y ≠ z ∧ z ∈ τ (X ∪ {y})
          ⟹  y ∉ τ (X ∪ {z})"
      proof (rule allI)
        fix z
        show "X ⊆ E ∧ y ∈ E - τ X ∧ z ∈ E - τ X ∧ y ≠ z ∧ z ∈ τ (X ∪ {y})  ⟹
            y ∉ τ (X ∪ {z})"
        proof (rule impI)
```

```
      assume z_y_prop: "X ⊆ E ∧ y ∈ E - τ X ∧ z ∈ E - τ X ∧ y ≠ z ∧ z ∈ τ (
        X ∪ {y})"
      show "y ∉ τ (X ∪ {z})"
      proof -
        let ?B = "E - τ X"
        have "X ⊆ E" using z_y_prop by simp
        then have "τ X ⊆ E" using ⟨set_system E F⟩ τ_in_E contains_empty_set
            by auto
        then have "?B ⊆ E" using ⟨X ⊆ E⟩ by simp
        have "finite E" using ⟨set_system E F⟩ unfolding set_system_def by auto
        then have "finite X" using ⟨X ⊆ E⟩ finite_subset by auto
        have "finite ?B" using ⟨?B ⊆ E⟩ ⟨finite E⟩ finite_subset by auto
        have "z ∈ ?B" using z_y_prop by simp
        let ?A = "E - τ (X ∪ {y})"
        have "y ∈ ?B" using z_y_prop by simp
        then have "y ∈ E" by simp
        then have "X ∪ {y} ⊆ E" using ⟨X ⊆ E⟩ by blast
        then have "?A ⊆ E" by simp
        have "finite (?A)" using ⟨finite E⟩ finite_subset by auto
        have "?B ∈ F" using τ_prop contains_empty_set ⟨set_system E F⟩ ⟨
            closed_under_union F⟩ ⟨τ X ⊆ E⟩ by (simp add: ⟨X ⊆ E⟩ assum1)
        have "τ (X ∪ {y}) ⊆ E" using ⟨X ∪ {y} ⊆ E⟩ τ_in_E ⟨set_system E F⟩
            contains_empty_set by simp
        then have "?A ∈ F" using ⟨closed_under_union F⟩ ⟨set_system E F⟩ τ_prop
          contains_empty_set ⟨X ∪ {y} ⊆ E⟩ by blast
        have τ_2nd_prop: "∀ X Y. X ⊆ E ∧ Y ⊆ E ∧ X ⊆ Y → τ X ⊆ τ Y" using
            τ_closure operator assum1 closure_operator.S_2 contains_empty_set
            ss_assum by blast
        have "X ⊆ X ∪ {y}" by auto
        then have "X ⊆ E ∧ (X ∪ {y}) ⊆ E ∧ X ⊆ X ∪ {y}" using ⟨X ⊆ E⟩ ⟨X ∪ {
            y} ⊆ E⟩ by auto
        then have "τ X ⊆ τ (X ∪ {y})" using τ_2nd_prop by blast
        then have "?A ⊆ ?B" by auto
        have "τ (X ∪ {y}) ⊆ E"
          using ⟨X ∪ {y} ⊆ E⟩ ⟨set_system E F⟩ τ_in_E contains_empty_set by
              auto
        have "y ∈ ?B" using z_y_prop by auto
        have "z ∈ ?B" using z_y_prop by auto
        have "z ∈ τ (X ∪ {y})" using z_y_prop by auto
        then have "z ∉ ?A"
          using ⟨z ∈ ?B⟩ by fastforce
        have "y ∈ X ∪ {y}" by simp
        have "∀ X. X ⊆ E → X ⊆ τ X" using τ_closure operator assum1
            closure_operator.S_1 contains_empty_set ss_assum by blast
        then have "X ∪ {y} ⊆ τ (X ∪ {y})" using ⟨X ∪ {y} ⊆ E⟩ by blast
        then have "y ∈ τ (X ∪ {y})" using ⟨y ∈ X ∪ {y}⟩ by auto
        then have "y ∉ ?A" by simp
        have "?A ⊆ ?B - {y,z}"
          using Diff_iff ⟨?A ⊆ ?B⟩ ⟨y \in τ (X ∪ {y})⟩ subset_Diff_insert
              subset_insert z_y_prop by auto
        have "card ?A < card ?B"
          by (metis ⟨?A ⊆ ?B⟩ ⟨finite ?B⟩ ⟨z ∈ ?B⟩ ⟨z ∉ ?A⟩ card_mono
```

```
                    card_subset_eq le_neq_implies_less)
              then have "∃x ∈ ?B - ?A. ?A ∪ {x} \in F"
                using ⟨?B ∈ F⟩ ⟨?A ∈ F⟩ ⟨greedoid E F⟩ greedoid.third_condition by
                    blast
              then obtain b where b_prop: "b ∈ ?B - ?A" "?A ∪ {b} ∈ F" by auto
```

The next listing is the proof of `?A ∪ B ⊈ E - (X ∪ {y})` by contradiction by applying τ operator on both sides, as in the informal proof, Theorem 3.5.

Listing 22: *Proof of ?A ∪ B ⊈ E - (X ∪ {y})*

```
have "¬?A ∪ {b} ⊆ E - (X ∪ {y})"
proof (rule ccontr)
    assume assum4: "¬ ¬?A ∪ {b} ⊆ E - (X ∪ {y})"
    then have "?A ∪ {b} ⊆ E - (X ∪ {y})" by auto
    then have "E - (?A ∪ {b}) ⊇ E - (E - (X ∪ {y}))" using in_E1 in_E2 by auto
    then have "E - (E - (X ∪ {y})) ⊆ E - (?A ∪ {b})" by simp
    then have ineq_one: "τ (E - (E - (X ∪ {y}))) ⊆ τ(E - (?A ∪ {b}))"
      by (meson Diff_subset τ_2nd_prop)
    have "E - (E - (X ∪ {y})) = X ∪ {y}" using ⟨X ∪ {y} ⊆ E⟩ by auto
    then have ineq_two: "τ (X ∪ {y}) ⊆ τ(E - (?A ∪ {b}))" using ineq\_one by simp
    have eq_one: "τ (X ∪ {y}) = E - ?A"
      by (metis Diff_partition Diff_subset_conv Un_Diff_cancel ⟨E - τ X ⊆ E⟩ ⟨X ∪
          {y} ⊆ E⟩ ⟨τ X ⊆ E⟩ ⟨set_system E F⟩ τ_in_E contains_empty_set
          double_diff)
    have "τ(E - (?A ∪ {b})) = E - (?A ∪ {b})" using b_prop(2) τ_prop2 ⟨set_system
        E F⟩ ⟨closed_under_union F⟩ contains_empty_set by simp
    then have "τ (X ∪ {y}) ⊆ E - (?A ∪ {b})" using ineq_two by simp
    then show "False" using eq_one b_prop(2)
      using assum4 b_prop2 by blast
qed
```

The next part, `b = y`, and the conclusion of the first implication is shown in the next listing.

Listing 23: *Proof of b = y and end of Accessibility $\Longrightarrow$ Antiexchange property*

```
  then have "b ∈ X ∪ {y}" using b_prop2 ⟨X ∪ {y} ⊆ E⟩ eqn using b_prop(1)
      insert_subset Diff_mono Un_insert_right ⟨X ∪ {y} ⊆ τ (X ∪ {y})⟩ equalityE
      sup_bot_right by auto
  then have "b = y" using ⟨b ∉ X⟩ by simp
  then have "?A ∪ {y} ∈ F" using b_prop(2) by auto
  have "y ∉ τ X" using ⟨y ∈ E - τ X⟩ by simp
  then have "y ∉ X" using ⟨X ⊆ τ X⟩ by auto
  then have prop2: "X ⊆ E - (?A ∪ {y})" using prop1 by auto
  have "z ∈ E - (E - τ (X ∪ {y}))" using ⟨z ∉ ?A⟩ ⟨?A ⊆ E⟩ using z_y_prop by auto
  then have "z ∈ E - (?A ∪ {y})" using ⟨y ≠ z⟩ by simp
  then have "(X ∪ {z}) ⊆ E - (?A ∪ {y})" using prop2 by simp
  then have "τ (X ∪ {z}) ⊆ τ (E - (?A ∪ {y}))" using τ_2nd_prop using ⟨X ⊆ E ∧ X
      ∪ {y} ⊆ E ∧ X ⊆ X ∪ {y}⟩ by auto
  then have "τ (X ∪ {z}) ⊆ (E - (?A ∪ {y}))" using ⟨(?A ∪ {y}) ∈ F⟩ τ_prop2 ⟨
      set_system E F⟩ ⟨closed_under_union F⟩ contains_empty_set by simp
  then show "y ∉ τ (X ∪ {z})" by auto
```

Now we set the proof structure for antiexchange property $\Longrightarrow$ accessibility by fixing `A`, `?X = E - A`, and properties about these sets. We obtain an element `a ∈ A` such that `τ(X ∪ {a})` is minimum.

This element exists and is obtained by lemma `min_card_exists`, which proves the existence using minimum cardinality of all τ(X ∪ {a}) such that a ∈ A.

Listing 24: *Start of Proof of Antiexchange property $\implies$ Accessibility*

```

show "antiexchange_property τ ⇒ accessible E F"
proof -
  assume "antiexchange_property τ"
  show "accessible E F"
  unfolding accessible_def
   proof (rule)
     show "set_system E F"
       using assms(2) by simp
     show "{} ∈ F ∧ (∀ X. X ∈ F - {{}} ⟶ (∃ x ∈ X. X - {x} ∈ F))"
     proof (rule)
       show "{} ∈ F" using assum2 by simp
       show "(∀ X. X ∈ F - {{}} ⟶ (∃ x ∈ X. X - {x} ∈ F))"
       proof (rule allI)
         fix A
         show "A ∈ F - {{}} ⟶ (∃ x ∈ A. A - {x} ∈ F)"
         proof (rule impI)
           assume "A ∈ F - {{}}"
           show "∃ x ∈ A. A - {x} ∈ F"
           proof  -
             let ?X = "E - A"
             have "A ∈ F" using ⟨A ∈ F - {{}}⟩ by simp
             have "τ (?X) = ?X" using ⟨A ∈ F⟩ ⟨set_system E F⟩ ⟨closed_under_union F⟩ assum2 τ_prop2 by simp
             have "A ≠ {}" using ⟨A ∈ F - {{}}⟩ by simp
             have "∃ a. a ∈ A ∧ ¬(∃ b. b ∈ A ∧ card (τ ((?X) ∪ {b})) < card (τ ((?X) ∪ {a})))"
               using min_card_exists ⟨A ∈ F - {{}}⟩ ⟨set_system E F⟩ ⟨closed_under_union F⟩ assum2 by auto
             then obtain a where a_prop: "a ∈ A ∧ ¬(∃ b. b ∈ A ∧ card (τ ((?X) ∪ {b})) < card (τ ((?X) ∪ {a})))"
               by auto
             then have "a ∈ A" by simp
             have a_prop2: "¬(∃ b. b ∈ A ∧ card (τ ((?X) ∪ {b})) < card (τ ((?X) ∪ {a})))" using a_prop by simp
             have "A ⊆ E" using ⟨A ∈ F - {{}}⟩ ⟨set_system E F⟩ unfolding set_system_def
               by simp
             then have "?X ⊆ E" by simp
             then have "?X ∪ {a} ⊆ E" using ⟨a ∈ A⟩ ⟨A ⊆ E⟩ by auto
             have "∀ X. X ⊆ E ⟶ X ⊆ τ X" using τ_closure_operator assum1(1) assum2 closure_operator.S_1 ss_assum by blast
             then have prop1: "?X ∪ {a} ⊆ τ ((?X) ∪ {a})" using ⟨(?X) ∪ {a} ⊆ E⟩ by blast
             have "τ (?X) ∪ {a} ⊆ E" using ⟨(?X) ∪ {a} ⊆ E⟩ τ_in_E ⟨set_system E F⟩ assum2 by simp
             have "finite E" using ⟨set_system E F⟩ unfolding set_system_def by simp
```

```
          then have "finite (τ (?X) ∪ {a})" using ⟨τ (?X) ∪ {a} ⊆ E⟩
            finite_subset by auto
```

We now prove the inclusion `τ(?X∪{a}) ⊆ ?X∪{a}` by contradiction, hence proving equality of the above two sets.

Listing 25: *Proof of τ(?X ∪ {a}) ⊆ ?X ∪ {a}*

```
 have "τ ((?X) ∪ {a}) ⊆ (?X) ∪ {a}"
proof (rule ccontr)
 assume assum6: "¬ (τ (?X ∪ {a}) ⊆ ?X ∪ {a})"
 show "False"
 proof -
   have "∃b. b ∈ τ (?X ∪ {a}) ∧ b ∉ (?X) ∪ {a} ∧ a ≠ b" using ⟨ (?X) ∪ {a} ⊆ τ
       (?X ∪ {a})⟩ using assum6 by auto
   then obtain b where b_prop: "b ∈ τ (?X ∪ {a})" "b ∉ ?X ∪ {a}" by auto
   then have "b ∈ E" using ⟨?X ∪ {a} ⊆ E⟩ ⟨set_system E F⟩ τ_in_E assum2 by blast
   then have "(?X ∪ {b}) ⊆ E" using ⟨?X ⊆ E⟩ by simp
   then have "τ (E - A ∪ {b}) ⊆ E" using τ_in_E ⟨set_system E F⟩ assum2 by simp
   then have "finite (τ (?X ∪ {b}))" using ⟨finite E⟩ finite_subset by simp
   have "b ∉ ?X" using b_prop(2) by simp
   then have "b ∉ τ ?X" using ⟨τ ?X = ?X⟩ by simp
   then have 1: "b ∈ E - τ ?X" using ⟨b ∈ E⟩ by simp
   have "b ∈ A" using ⟨b ∉ ?X⟩ ⟨A ⊆ E⟩
     using ⟨b ∈ E⟩ by blast
   have 2: "b ≠ a" using b_prop(2) by simp
   have "E - ?X = A" by (simp add: ⟨A ⊆ E⟩ double_diff)
   then have "E - τ (?X) = A" using ⟨τ ?X = ?X⟩ by simp
   then have "a ∈ E - τ (?X)" using a_prop by simp
   then have fact_one: "?X ⊆ E ∧ a ∈ E - τ ?X ∧ b ∈ E - τ ?X ∧ a ≠ b ∧ b ∈ τ (?X
        ∪ {a})" using
       ⟨E - A ⊆ E⟩ 1 2 b_prop(1) by simp
   then have "a ∉ τ ((?X) ∪ {b})" using ⟨antiexchange_property τ⟩
     unfolding antiexchange_property_def by simp
   have "τ ((?X) ∪ {a}) ⊆ E" using ⟨(?X) ∪ {a} ⊆ E⟩ τ_in_E ⟨set_system E F⟩ assum2
        by simp
   then have "τ ((?X) ∪ {a}) ∪ {b} ⊆ E" using ⟨b ∈ E⟩ by simp
   have ⟨finite (τ ((?X) ∪ {a}))⟩ using ⟨τ(?X ∪ {a}) ⊆ E⟩ ⟨finite E⟩ finite_subset
       by auto
   have "a ∉ ?X" using ⟨a ∈ A⟩ by simp
   then have "(?X) ⊆ (?X) ∪ {a}" by auto
   then have "τ(?X) ⊆ τ ((?X) ∪ {a})" using ⟨?X ⊆ E⟩ ⟨(?X) ∪ {a} ⊆ E⟩
     using ⟨τ (?X) = ?X⟩ prop1 by auto
   then have "(?X) ⊆ τ ((?X) ∪ {a})" using ⟨τ(?X) = ?X⟩ by simp
   then have "(?X) ∪ {b} ⊆ τ ((?X) ∪ {a}) ∪ {b}" using ⟨b ∉ ?X⟩
     by blast
   then have "τ((?X) ∪ {b}) ⊆ τ (τ ((?X) ∪ {a}) ∪ {b})" using ⟨(?X) ∪ {b} ⊆ E⟩ ⟨τ
       ((?X) ∪ {a}) ∪ {b} ⊆ E⟩
     by (meson τ_closure_operator assum1(1) assum2 closure_operator_def ss_assum)
   also have "... = τ (τ ((?X) ∪ {a}))" using b_prop(1)
     by (simp add: insert_absorb)
   also have "... = τ ((?X) ∪ {a})" using ⟨τ ((?X) ∪ {a}) ⊆ E⟩
     by (metis ⟨?X∪{a} ⊆ E⟩ τ_closure_operator assum1 closure_operator.S_3 assum2)
   finally have "τ ((?X) ∪ {b}) ⊆ τ ((?X) ∪ {a})" by simp
```

```
    have "τ ((?X) ∪ {b}) ≠ τ ((?X) ∪ {a})"
    proof
      assume "τ ((?X) ∪ {b}) = τ ((?X) ∪ {a})"
      show "False"
      proof -
        have "a ∈ ?X ∪ {a}" by simp
        then have "a ∈ τ (?X ∪ {a})" using ⟨?X ∪ {a} ⊆ τ (?X ∪ {a})⟩ by simp
        then show ?thesis using ⟨a ∉ τ(?X ∪ {b})⟩ using ⟨τ (E - A ∪ {b}) = τ (E - A
            ∪ {a})⟩ by auto
      qed
    qed
    then have "card (τ ((?X) ∪ {b})) < card (τ ((?X) ∪ {a}))"
      using ⟨finite (τ ((?X) ∪ {a}))⟩ by (meson ⟨τ (E - A ∪ {b}) ⊆ τ (E - A ∪ {a})⟩
          card_mono card_subset_eq le_neq_implies_less)
    then have "b ∈ A ∧ card (τ (?X ∪ {b})) < card (τ (?X ∪ {a}))" using ⟨b ∈ A⟩ by
        simp
    then show "False" using a_prop2 by auto
  qed
  qed
```

The first part of the above listing deals by assuming on the contrary that `τ(?X ∪ {a})` $\not\subseteq$ `?X ∪ {a}`. We then obtain an element `b` in the difference of the sets. By the forward reasoning explained in Theorem 3.5, we obtain `τ(?X ∪ {b}) ⊆ τ?X ∪ {a}`. A small subproof shows that this inclusion is strict, as `a` is not in the set `τ(?X ∪ {b})` but in the set `τ(?X ∪ {a})`. This disproves the choice of `a`, and `τ(?X ∪ {a}) = ?X ∪ {a}`.

The last listing for this theorem shows the accessibility property from the above proved equation.

Listing 26: *End of Antiexchange Property* $\implies$ *Accessibility*

```
then have eqn: "τ (?X ∪ {a}) = ?X ∪ {a}" using prop1 by auto
have set_in_F: "E - τ (?X ∪ {a}) ∈ F" using ⟨?X ∪ {a} ⊆ E⟩ ⟨set_system E F⟩ ⟨
    closed_under_union F⟩ τ_prop assum2 by simp
have "E - (?X ∪ {a}) = A - {a}" using ⟨A ⊆ E⟩ by fastforce
then have "E - (τ (?X ∪ {a})) = A - {a}" using eqn by simp
then have "A - {a} ∈ F" using set_in_F by simp
then show ?thesis using a_prop(1) by auto
qed
```

The proof in [KV06] does not explicity use auxiliary lemmas such as Theorem 3.2, Theorem 3.3 and Theorem 3.4. As much as the fundamental idea of the proof is taken from [KV06], these auxiliary lemmas are proved separately to strengthen the understanding of the operator τ and conveniently use them in various parts of the main proof.

4 An Example of a Greedoid and the Greedy Algorithm

4.1 Theory: An Example of a Greedoid and the Greedy Algorithm

4.1.1 An Example of a Greedoid

Definition 4.1. ([Deo16], Section 9.1) A digraph consists of a finite set of vertices V and edges E such that $E \subseteq V \times V$.

Definition 4.2. ([Deo16], Section 9.11) An acyclic digraph is a digraph without any cycles, that is, there is no path $p \subseteq V$ (where each $(p_i, p_{i+1}) \in E$, $\forall\, i < |p|$) such that for any $v \in V$, $v = p_1$ and $v = p_{|p|}$.

Definition 4.3. ([Deo16], Section 9.4) A weakly connected digraph is a digraph such that a path exists between any two vertices of the digraph, that is, $\forall\, v,\, u \in V$ we have a set $p \subseteq V$ such that $u = p_1$ and $v = p_{|p|}$ (or vice versa) and each (p_i, p_{i+1}) or $(p_{i+1}, p_i) \in E$, $\forall\, i < |p|$).

The notion of strongly connected digraphs are not discussed in this report. Therefore any references made to connected digraphs mean weakly connected digraphs only.

Definition 4.4. ([Deo16], Section 9.6) A directed tree is a digraph that is acyclic, connected and satisfies the relation $|V| = |E| + 1$.

The next theorem is proved for arborescences (a specific kind of directed tree) in [KV06]. However, the proof here is written for directed trees (the general case of arborescences), as the formalization verfies the proof for general directed trees.

Theorem 4.1. ([KV06], Proposition 14.6) Let $(V,\, E)$ be a digraph and fix a vertex $r \in V$. Let $\mathcal{F}$ be the edge set of all trees containing r. Then, $(E,\, \mathcal{F})$ is a greedoid.

Proof. Since V is finite, we can say that E is also finite. Hence, by definition of $\mathcal{F}$, $(E,\, \mathcal{F})$ is a set system.
Now consider the graph $(\{r\}\,,\, \{\})$. It is acyclic and connected as r is the only vertex present. Moreover, $|\,\{\}\,| = |\,\{r\}\,| - 1 = 1 - 1 = 0$. Hence, $(\{r\}\,, \{\})$ is a tree containing r, and $\{\} \in \mathcal{F}$. This proves axiom 1 of greedoids. (Definition 2.1)
Now, consider any two trees $(V_1,\, X)$ and $(V_2,\, Y)$, both containing r such that $|Y| < |X|$. Then $|V_2| + 1 < |V_1| + 1 \implies |V_2| < |V_1|$. Now, consider a vertex $x \in V_1 - V_2$. As V_1 contains r and is the vertex set of a tree, it is connected. Hence we can find a path from x and $r \in V_1$. Let this path be p. Now, since $x \in V_1$ but not in V_2, we can find a vertex in this path that lies in V_1 but not V_2 such that the vertex before it in that path is in $V_1 \cap V_2$. We prove the existence of this vertex by case analysis.
Case 1. $\forall\, i < |p|$, $p_i \in V_1 \cap V_2$: In this case, we know that $p_{|p|} = x$, $x = p_{|p|} \in V_1 - V_2$, and $p_{|p|-1} \in V_1 \cap V_2$ by assumption. Hence the required vertex is $p_{|p|} = x$.
Case 2. $\exists\, i < |p|$, $p_i \notin V_1 \cap V_2$. In this case, we obtain an i such that $i < |p|$, $p_i \notin V_1 \cap V_2$. Splitting V_1 as $V_1 = (V_1 \cap V_2) \cup (V_1 - V_2)$, we have, $p_i \in V_1 - V_2$. Let $A = \{j \,|\, j < |p| \wedge p_j \in V_1 - V_2\}$. This set is finite by definition. Also it contains i by the properties of obtained i. Therefore we can find a minimum k in A and p_k becomes our required vertex. We prove this statement by contradiction. Assuming $p_{k-1} \notin V_1 \cap V_2$, we have $p_{k-1} \in V_1 - V_2$, but this disproves the fact that k is minimum.

A similar proof follows for when $p_1 = x$ and $p_{|p|} = r$. We now have an element $p_i \in V_1 - V_2$ (say w) and $p_{k-1} \in V_1 \cap V_2$ (say v). We observe that $(p_{k-1},\, pk)$ (or (p_k, p_{k-1})) $\in X$ but not in Y. Our claim now is that $((V_2 \cup \{w\}),\, (Y \cup \{(v, w)\}))$ is a tree containing r. A similar proof follows for $((V_2 \cup \{w\}),\, (Y \cup \{(w, v)\}))$.
Since, $(Y \cup \{(v, w)\}) \subseteq (V_2 \cup \{w\}) \times (V_2 \cup \{w\})$, the above set system is a digraph containing r $(r \in V_2)$.

We prove the acyclic property by contradiction. We assume there exists a cycle p at some vertex $v' \in (V_2 \cup \{w\})$. Now, proceeding by case analysis:
Case 1. $v' = w$: If we have a cycle at vertex w, we have $p_1 = w$ and $p_2 \in V_2$. Then we have $(w,\, p_2) \in (Y \cup \{(v, w)\})$. But this is a contradiction as we have no edge of the form $(w,\, v'')$ for all $v'' \in V_2$.
Case 2. $v' \neq w$: If we have a cycle at vertex v' that is not w, then v' is in V_2. Then we split the proof again by case analysis on $w \in p$:
Subcase (i): $w \in p$: If this is true, then we can assume $p_i = w$ for some $i < |p|$. i cannot be equal to $|p|$ as we already know that $p_{|p|} = v'$ by definition of a cycle. Now, we have $(p_i,\, p_{i+1}) \subseteq (Y \cup \{(v, w)\})$. But this is a contradiction as there exists no edge of the form $(w,\, v'')$ for all $v'' \in V_2$.
Subcase (ii): $w \notin p$: This implies that all vertices in the path p are in V_2, implying that we have found an cycle in $(V_2,\, Y)$. This is a contradiction as per our assumption of $(V_2,\, Y)$ being a tree.

Now, the digraph $((V_2 \cup \{w\}),\, (Y \cup \{(v, w)\}))$ is connected, because for all $v' \in V_2$ we can find a path p from v' to v (since $(V_2,\, Y)$ is a tree). Appending the vertex w in this path, we obtain a new path p'. This is a valid path as $\forall\, i < |p'| - 1,\, (p'_i,\, p_{i+1})$ (or vice versa) $\in (Y \cup \{(v, w)\})$ by properties of p and for $i = |p'| - 1,\, (p'_i,\, p_{i+1}) = (v,\, w) \in (Y \cup \{(v, w)\})$. Also, we have $|V_2| = Y + 1 \implies (|V_2| + 1) = (Y + 1) + 1 \implies |(V_2 \cup \{w\})|, = |(Y \cup \{(v, w)\})| + 1$. Therefore, $((V_2 \cup \{w\}),\, (Y \cup \{(v, w)\}))$ is a tree and $(Y \cup \{(v, w)\})) \in \mathcal{F}$. We have now found an element $(v,\, w) \in X$ but not in Y such that $(Y \cup \{(v, w)\})) \in \mathcal{F}$, proving the second axiom for greedoids. (Definition 2.1) □

4.1.2 The Greedy Algorithm

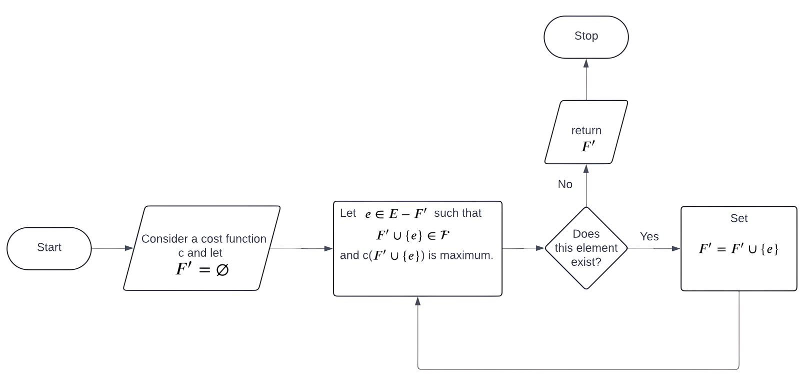
Definition 4.5. ([KV06], Section 2.1.) A cost (or weight) function is a function $c : E \to \mathbb{R}$ that has the following behavior:
1. $c(\emptyset) = 0$.
2. $c(X) = \sum_{e \in X} c(e)$.
This function extends to a modular cost (or weight) function when $c : 2^E \to \mathbb{R}$.

Greedy algorithms focus on finding an overall optimal solution by finding locally optimal solutions at each step. The greedy algorithm for greedoids takes a function, the cost function c and a set F (a subset of E), proceeds to find the next element y in $E - F$ such that $F \cup \{y\} \in \mathcal{F}$ and $c(F' \cup \{y\})$ is maximum, given a greedoid $(E, \mathcal{F})$. When no such element exists, the algorithm stops and returns F, else we repeat the search for the required element for $F \cup \{y\}$.

This function terminates as the cardinality of number of elements that serve as best candidates reduce as it goes through each recursive call.

The flowchart below describes the workflow of the greedy algorithm.

Figure 1: Greedy Algorithm for a Greedoid $(E, \mathcal{F})$



The above algorithm is shown to return an optimum solution for weight functions and greedoids that satisfy the strong exchange property.

Definition 4.6. ([KV06], Theorem 14.7.) A set system $(E, \mathcal{F})$ satisfies the strong exchange property if and only if for all A, $B \in \mathcal{F}$, B is maximal in $\mathcal{F}$, $A \subseteq B$ and $x \in E - B$ such that $A \cup \{x\} \in \S$, then $\exists y$ such that $A \cup \{y\} \in \mathcal{F}$ and $(B - \{y\}) \cup \{x\} \in \mathcal{F}$.

Theorem 4.2. ([KV06], Theorem 14.7) For all modular weight functions c, the Greedy Algorithm for Greedoids gives an optimum solution if and only if the greedoid satisfies the strong exchange property.

4.2 Formalization: An Example of a Greedoid and the Greedy Algorithm

4.2.1 An Example of a Greedoid

We begin the formalization of the explained example of a greedoid by defining a locale that fixes the set of vertices and edges. The definitions of digraph, acyclic, weakly connected and directed tree are then mentioned.

Listing 27: *Greedoid Example Start*

```
locale greedoid_example  =
fixes V :: "'a set"          (* Set of vertices *)
and E :: "('a × 'a) set" (* Set of directed edges *)
  assumes finite_assum_V: "finite V"

context greedoid_example
begin

definition digraph::"'a set ⇒ ('a × 'a) set ⇒ bool" where "digraph F G = (G ⊆
    F × F)"

definition acyclic::"'a set ⇒ ('a × 'a) set ⇒ bool" where "acyclic F G = ((G =
    ∅) ∨ (∀ v ∈ F. ¬ (∃ p. p ≠ ∅ ∧ hd p = v ∧ last p = v ∧ (∀ i < length p −
    1. (p ! i, p ! (i + 1)) ∈ G) ∧ length p ≥ 2)))"

definition connected::"'a set ⇒ ('a × 'a) set ⇒ bool" where "connected F G =
    ((G = ∅ ∧ card F = 1) ∨ (∀ u v. u ∈ F ∧ v ∈ F ∧ u ≠ v ⟶ (∃ p. p ≠ ∅ ∧
    ((hd p = v ∧ last p = u) ∨ (hd p = u ∧ last p = v)) ∧ (∀ i < length p − 1.
     ((p ! i, p ! (i + 1)) ∈ G) ∨ (p ! (i + 1), p ! i) ∈ G) ∧ length p ≥ 2)))"

definition tree::"'a set ⇒ ('a × 'a) set ⇒ bool" where "tree F G ⇔ digraph F
    G ∧ acyclic F G ∧ connected F G ∧ (card G = card F − 1)"
```

We now begin the proof of Theorem 4.1. The proof method used is `unfold_locales` which sets the subgoals of the proof as the assumptions of the `greedoid` locale. The goals `{} ∈ ?F` and `set_system E ?F` are proved by unfolding defintions.

Listing 28: *Proof of First Two Greedoid Assumptions*

```
lemma greedoid_graph_example: assumes "digraph V E" "r ∈ V"
shows "greedoid E {Y. ∃ X. X ⊆ V ∧ Y ⊆ E ∧ r ∈ X ∧ tree X Y}"
proof (unfold_locales)
    let ?F = "{Y. ∃ X. X ⊆ V ∧ Y ⊆ E ∧ r ∈ X ∧ tree X Y}"
    show first_part: "{} ∈ {Y. ∃ X. X ⊆ V ∧ Y ⊆ E ∧ r ∈ X ∧ tree X Y}"
    proof -
        have factone: "{r} ⊆ V" using assms(2) by simp
```

```
          have "tree {r} {}" unfolding tree_def
          proof
            show "digraph {r} {}" unfolding digraph_def by auto
            have 1: "acyclic {r} {}" unfolding acyclic_def by simp
            have 2: "connected {r} {}" unfolding connected_def by simp
            have 3: "card {} = card {} - 1" by simp
            then show "local.acyclic {r} {} ∧ local.connected {r} {} ∧ card {}
              = card {r} - 1" using 1 2 by simp
          qed
    then have "{r} ⊆ V ∧ {} ⊆ E ∧ r ∈ {r} ∧ tree {r} {}" using factone by simp
    then show ?thesis by blast
  qed
  show "set_system E ?F" unfolding set_system_def
  proof
    show "finite E"
    proof -
      have "E ⊆ V × V" using assms(1) unfolding digraph_def by simp
      then show "finite E" using finite_assum_V
        by (simp add: finite_subset)
    qed
    show "∀Z∈?F. Z ⊆ E" by auto
  qed
```

The next listing starts the proof of axiom 2 of greedoids from Definition 2.1.

Listing 29: *Start of the Proof of Third Greedoid Assumption*

```
  show "∀X Y. X ∈ ?F ∧ Y ∈ ?F ∧ card Y < card X ⟹ ∃x∈X - Y. Y ∪ {x} ∈ ?F"
  proof -
    fix X Y
    assume assum2: "X ∈ ?F ∧
         Y ∈ ?F ∧ card Y < card X"
    then obtain V1 where V1_prop: "V1 ⊆ V ∧ X ⊆ E ∧ r ∈ V1 ∧ tree V1 X" by auto
    then have "tree V1 X" by simp
    then have V1_card: "card V1 - 1 = card X" unfolding tree_def by auto
    have "V1 ≠ {}" using V1_prop by auto
    have "finite V1" using V1_prop finite_assum_V finite_subset by auto
    then have "card V1 > 0" using ⟨V1 ≠ {}⟩ by auto
    then have factone: "card V1 = card X + 1" using V1_card by auto
    obtain V2 where V2_prop: "V2 ⊆ V ∧ Y ⊆ E ∧ r ∈ V2 ∧ tree V2 Y" using assum2
       by auto
    then have "tree V2 Y" by simp
    then have V2_card: "card V2 - 1 = card Y" unfolding tree_def by auto
    have "V2 ≠ {}" using V2_prop by auto
    have "finite V2" using V2_prop finite_assum_V finite_subset by auto
    then have "card V2 > 0" using ⟨V2 ≠ {}⟩ by auto
    then have facttwo: "card V2 = (card Y) + 1" using V2_card by auto
    have "card Y < card X" using assum2 by simp
    then have "card Y + 1 < card X + 1" by simp
    then have "card V2 < card V1" using factone facttwo by simp
    have "X = (X - Y) ∪ (X ∩ Y)" by auto
    show "∃x∈X - Y. Y ∪ {x} ∈ ?F"
    proof -
      have "V1 ≠ {}" using V1_prop by auto
```

```
        then have "V1 ≠ V2" using ⟨card V2 < card V1⟩ by auto
        then have "V1 - V2 \neq {}"
          by (metis ⟨card V2 < card V1⟩ ⟨finite V1⟩ ⟨finite V2⟩ bot.extremum_strict
              bot_nat_def card.empty card_less_sym_Diff)
        then obtain x where "x ∈ V1 - V2" by auto
        then have "x ∉ V2" by simp
        have "x ∈ V1" using ⟨x ∈ V1 - V2⟩ by simp
        have "r ∈ V2" using V2_prop by simp
        then have "x ≠ r" using ⟨x ∉ V2⟩ by auto
        have "X ≠ {}"
          using ⟨card Y < card X⟩ card.empty by auto
        have "r ∈ V1" using V1_prop by simp
        have "tree V1 X" using V1_prop by simp
        then have "connected V1 X" unfolding tree_def by simp
        then have "(X = {} ∧ card V1 = 1) ∨ (∀u v. u ∈ V1 ∧ v ∈ V1 ∧ u ≠ v ⟶ (
            ∃p. p ≠ {} ∧ ((hd p = u ∧ last p = v ) ∨ (hd p = v ∧ last p = u)) ∧
            (∀i < length p - 1. ((p  i, p  (i + 1)) ∈ X) ∨ ((nth p (i + 1), nth p
            i) ∈ X)) ∧ length p ≥ 2))" unfolding
            connected_def by auto
        then have "(∀u v. u ∈ V1 ∧ v ∈ V1 ∧ u \neq v ⟶ (∃p. p ≠ {} ∧ ((hd p =
            u ∧ last p = v ) ∨ (hd p = v ∧ last p = u)) ∧ (∀i < length p - 1. ((p
             i, p  (i + 1)) ∈ X) ∨ ((nth p (i + 1), nth p i) ∈ X)) ∧ length p ≥
            2))" using ⟨X ≠ {}⟩ by simp
        then have "(∃p. p ≠ {} ∧ ((hd p = r ∧ last p = x) ∨ (hd p = x ∧ last p =
            r)) ∧ (∀i < length p - 1. ((p  i, p  (i + 1)) ∈ X) ∨ (nth p (i + 1),
            nth p i) ∈ X) ∧ length p ≥ 2))" using ⟨r ∈ V1⟩ ⟨x ∈ V1⟩ ⟨x ≠ r⟩ ⟨X ≠
            {}⟩ by simp
        then obtain p where p_prop: "(p ≠ {} ∧ ((hd p = r ∧ last p = x) ∨ (hd p =
             x ∧ last p = r)) ∧ (∀i < length p - 1. ((p  i, p  (i + 1)) ∈ X) ∨ (
            nth p (i + 1), nth p i) ∈ X) ∧ length p ≥ 2))" by auto
```

The above listing fixes two variables `X, Y` in the edge set `?F` of consideration. We assume `card X > card Y` and obtain the respective set of vertices, `V1` and `V2`. Since `V2` is nonempty, and `card X < card Y`, we can conclude that `card V1 > card V2` and can obtain a vertex `x ∈ V2 ∧ x ∉ V1`. Since `tree V1 X`, we have a path `p` between `r` and `x`.

The next three listings obtains variables `v', w` such that `v' ∈ V1 ∩ V2, w ∈ V1 - V2` and `(v', w) ∈ X` or `(w, v') ∈ X`. The proof of obtaining these variables is done by case analysis of `hd p = r ∧ last p = x`.

For the first case, we start by proving a property of path `p` - every element of `p` lies in `V1`. This is done by case analysis on indices of `p`.

Listing 30: *Proof of Property of path p*

```
  have "∃v' w. v' ∈ V1 ∩ V2 ∧ w ∈ V1 - V2 ∧ ((v', w) ∈ X ∨ (w, v') ∈ X)"
        proof (cases "hd p = r ∧ last p = x")
          case True
          then have "last p = x" by simp
        have "hd p = r" using True by simp
        have "length p ≥ 2" using p_prop by simp
        have p_prop2: "∀i < length p - 1. ((p  i, p  (i + 1)) ∈ X) ∨ (nth p (i +
            1), nth p i) ∈ X" using p_prop by simp
        have "digraph V1 X" using ⟨tree V1 X⟩ unfolding tree_def by simp
```

```
        then have "X ⊆ V1 × V1" unfolding digraph_def by simp
        have p_prop3: "∀i ≤ length p - 1. (nth p i) ∈ V1"
        proof (rule allI)
          fix i
          show "i ≤ length p - 1 ⟹ p  i ∈ V1"
          proof (cases "i = length p - 1")
            case True
            then have "i ≠ 0" using ⟨length p ≥ 2⟩ by simp
            then have "i - 1 = length p - 1 - 1" using True by simp
            then have "i = (i - 1) + 1" using ⟨i \neq 0⟩ by simp
            then have "i - 1 < length p - 1" using ⟨length p ≥ 2⟩ ⟨i - 1 = length
                p - 1 - 1⟩ by simp
            then have "(nth p (i - 1), (nth p i)) ∈ X ∨ (nth p i, nth p (i - 1))
                ∈ X" using p\_prop ⟨i = (i - 1) + 1⟩ by auto
            then have "(nth p (i - 1), (nth p i)) ∈ V1 × V1" using ⟨X ⊆ V1 × V1⟩
                by auto
            then show ?thesis using True by auto
          next
            case False
            show "i ≤ length p - 1 ⟹ (nth p i) ∈ V1"
            proof
              assume "i ≤ length p - 1"
              then have "i < length p - 1" using ⟨length p ≥ 2⟩ False by simp
              then have "(nth p i, (nth p (i + 1))) ∈ X ∨ (nth p (i + 1), nth p i
                ) ∈ X" using p_prop by simp
              then have "(nth p i, (nth p (i + 1))) ∈ V1 × V1" using ⟨X ⊆ V1 × V1
                ⟩ by auto
              then show "(nth p i) ∈ V1" by simp
          qed
        qed
      qed
        have "V1 = (V1 ∩ V2) ∪ (V1 - V2)" by auto
        then have V1_el_prop: "∀v. v ∈ V1 ⟹ v ∈ (V1 - V2) ∨ v ∈ (V1 ∩ V2)" by
            auto
```

Now, we show the proof of the existence of an index `i` in path `p` such that `(nth p i) ∈ V1 ∩ V2`, `(nth p (i - 1)) ∈ V1 - V2` and `(nth p (i - 1), nth p i) ∈ X ∨ (nth p i, nth p (i - 1)) ∈ X`. The proof of this is case analysis on the elements of `p` - we split the proof into two cases depending on whether all elements of `p` minus the last one lie in `V1 ∩ V2`. The first case is when the statement is true- if all elements of `p` minus the last one lie in `V1 ∩ V2`. The claim is true as we can take `i = length p - 1`. This is because `(nth p (length p - 1)) = x ∈ V1 - V2` and `(nth p (length p - 1 - 1)) ∈ V1 ∩ V2` by assumption. The above index satisfies the edge property as well.
The proof of the second case is slightly nontrivial.

Listing 31: *Second Case of Proof of Existence of Index i*

```
  next
case False
then have "∃i < (length p) - 1. (nth p i) ∉ V1 ∩ V2" by auto
then obtain i where i_prop: "i < (length p) - 1 ∧ (nth p i) ∉ V1 ∩ V2" by auto
then have i_prop2: "(nth p i) ∉ V1 ∩ V2" by simp
then have "i < length p - 1" using i_prop by simp
then have "(nth p i) ∈ V1 - V2" using V1_el_prop i_prop2 p_prop3 by simp
```

```
  let ?A = "{j. j < length p - 1 ∧ (nth p j) ∈ V1 - V2}"
  have "finite ?A" by simp
  have "i ∈ ?A" using ⟨i < length p - 1⟩ ⟨(nth p i) ∈ V1 - V2⟩ by simp
  then have "?A ≠ {}" by auto
  then have "Min ?A ∈ ?A" using ⟨finite ?A⟩ Min_in by blast
  let ?k = "Min ?A"
  have min_prop: "∀j. j ∈ ?A ⟶ ?k ≤ j" by simp
  have k_prop: "?k < length p - 1 ∧ (nth p ?k) ∈ V1 - V2" using ⟨?k ∈ ?A⟩ by simp
  have "(nth p 0) = r" using p_prop hd_conv_nth True by metis
  then have "(nth p 0) ∈ V1 ∩ V2" using V1_prop V2_prop by simp
  have "(nth p ?k) ∈ V1 - V2" using k_prop by simp
  then have "?k ≠ 0" using ⟨(nth p 0) ∈ V1 ∩ V2⟩
    by (metis DiffD2 ⟨p ! 0 = r⟩ ⟨r ∈ V2⟩)
  then have "?k - 1 < ?k" by simp
  have "?k - 1 < length p - 1" using k_prop by auto
  have k_prop4: "(nth p (?k - 1)) ∈ V1 ∩ V2"
  proof (rule ccontr)
    assume "(nth p (?k - 1)) ∉ V1 ∩ V2"
    then have "(nth p (?k - 1)) ∈ V1 - V2" using p_prop3 V1_el_prop i_prop2 ⟨?k - 1
        < length p - 1⟩ by simp
    then have "?k - 1 < length p - 1" using ⟨?k - 1 < ?k⟩ k_prop by simp
    then have "?k - 1 ∈ ?A" using ⟨(nth p (?k - 1)) ∈ V1 - V2⟩ by simp
    then show "False" using min_prop ⟨?k - 1 < ?k⟩
      using less_le_not_le by blast
  qed
  then have k_prop3: "?k ≤ length p - 1 ∧ (nth p ?k) ∈ V1 - V2" using k_prop by
      simp
  then show ?thesis using k_prop4 by auto
qed
```

This listing deals with the case when not all elements of `p` minus the last one lie in `V1 ∩ V2`. This means we can obtain an index `i` where `(nth p i) ∈ V1 - V2`. We take the set of all such `i`s and observe that the minimum of this set is the required index.
Having obtained these indices, we see that `?v = v' = (nth p i)`, `?v = w = (nth p (i - 1))` and they satisfy the edge conditions as well. The proof for the second case, that is, `¬hd p = r ∧ last p = x` is skipped as it is analogous to the case discussed.

We observe that both edges `(?v, ?w)`, `(?w, ?v)` don't lie in `Y` as `?w ∉ V2`. Hence, we have obtained an element, say, `(v', w) ∈ X - Y`. Our goal is to prove that `Y ∪ {(v', w)} ∈ ?F`.

When `Y = {} ∧ card V2 = 1`, `V2 = {r}`, `?v = r` and `Y ∪ {(?v, ?w)} =` When `Y = {} ∧ card V2 = 1`, `V2 = {r}`, `?v = r`,`Y ∪ {(?v, ?w)} = {(r, ?w)}` and `V2 ∪ {?w} = {r, ?w}`. In this case, `(V2 ∪ {?w}) (Y ∪ {(v', w)})` is a tree containing `r` by triviality and the correspoding edge set is a greedoid.

We prove the case for when `¬(Y = {} ∧ card V2 = 1)`. The proofs for digraph and cardinality follow as `tree V2 Y` is true.

To prove weakly connected, we do a double case analysis on `v ∈ V2` and `u ∈ V2`. When both are true, we can obtain a path between them as `V2 Y` is weakly connected.

When `u = ?w`, we split the proof again into two cases, when `v = ?v` and otherwise. If the latter is true, then the required path is `(?v, ?w)`. The next listing shows the second case - when `v ≠ ?v`. We find a path from `v` to `?v` since `V2 Y` is connected. Then we append `?w` to that path, to prove that `?w` is connected to the rest of `V2`.

Listing 32: *A Snippet from the proof of Connected Property*

```
  next
case False
then have "u = ?w" using assm4 by simp
show ?thesis
proof (cases "v ≠ ?v")
  case True
  then have "(Y = {} ∧ card V2 = 1) ∨ (∃q. q ≠ [] ∧ ((hd q = v ∧ last q = ?v) ∨
      (hd q = ?v ∧ last q = v)) ∧ (∀ia < length q - 1. ((q ! ia, q ! (ia + 1)) ∈
      Y) ∨ (nth q (ia + 1), nth q ia) ∈ Y) ∧ length q ≥ 2)" using ⟨v ∈ V2⟩
    ⟨?v ∈ V1 ∩ V2⟩ Y_connected by auto
  then have "(∃q. q ≠ [] ∧ ((hd q = v ∧ last q = ?v) ∨ (hd q = ?v ∧ last q = v))
       ∧ (∀ia < length q - 1. ((q ! ia, q ! (ia + 1)) ∈ Y) ∨ (nth q (ia + 1), nth
       q ia) ∈ Y) ∧ length q ≥ 2)" using ⟨¬ (Y = {} ∧ card V2 = 1)⟩ by auto
  then obtain q where q_prop: "q ≠ [] ∧ ((hd q = v ∧ last q = ?v) ∨ hd q = ?v ∧
      last q = v) ∧ (∀ia < length q - 1. ((q ! ia, q ! (ia + 1)) ∈ Y) ∨ (nth q (
      ia + 1), nth q ia) ∈ Y) ∧ length q ≥ 2" by auto
  show ?thesis
  proof (cases "hd q = v ∧ last q = ?v")
    case True
    then have "¬(hd q = ?v ∧ last q = v)" using ⟨v ≠ ?v⟩ by simp
    let ?q = "q @ [?w]"
    have "q ≠ [] ∧ last q = ?v" using q_prop True by simp
    then have "(nth q (length q - 1)) = ?v" using q_prop last_conv_nth
      by fastforce
    then have v_cont: "(nth ?q (length ?q - 1 - 1)) = ?v"  by (metis
        Cons_eq_appendI ⟨q ≠ [] ∧ last q = ?v⟩ append.assoc append_butlast_last_id
         butlast_snoc length_butlast nth_append_length)
    have "length ?q ≥ 2" using q_prop by simp
    then have length_prop: "(length ?q - 1 - 1) + 1 = length ?q - 1 " by simp
    have "(nth ?q (length ?q - 1)) = ?w" by auto
    then have last_edge: "(nth ?q (length ?q - 1 - 1), nth ?q ((length ?q - 1 - 1)
         + 1 )) = (?v, ?w)" using v_cont length_prop by simp
    have "?q ≠ []" using q_prop by simp
    have hd_last_q: "hd ?q = v ∧ last ?q = ?w" using q_prop True by auto
    have "length ?q - 1 - 1 < length ?q - 1" using ⟨length ?q ≥ 2⟩ by simp
    have "∀i < length ?q - 1. ((?q ! i, ?q ! (i + 1)) ∈ Y ∪ {(?v, ?w)} ∨ (nth ?q
        (i + 1), nth ?q i) ∈ Y ∪ {(?v, ?w)})"
    proof -
      have "∀i < length q - 1. (nth ?q i) = (nth q i)"
        by (metis ⟨q ≠ [] ∧ last q = v'⟩ add_lessD1
            canonically_ordered_monoid_add_class.lessE diff_less
            length_greater_0_conv nth_append zero_less_one)
      moreover have "length ?q - 1 - 1 = length q - 1" by simp
      ultimately have "∀i < length ?q - 1 - 1. (nth ?q i) = (nth q i)" by simp
      then have q_el_prop: "∀i < length ?q - 1 - 1. (nth ?q i, nth ?q (i + 1)) ∈ Y
           ∪ {(?v, ?w)} ∨ (nth ?q (i + 1), nth ?q i) ∈ Y ∪ {(?v, ?w)}" using
          q_prop ⟨length ?q - 1 -1 = length q - 1⟩
        by (metis UnI1 less_diff_conv nth_append)
```

```
      have "(nth ?q (length ?q - 1 - 1), nth ?q (length ?q - 1)) = (?v, ?w)" using
           v_cont by auto
      then have "(nth ?q (length ?q - 1 - 1), nth ?q (length ?q - 1)) ∈ Y ∪ {(?v,
           ?w)}" by simp
      then have "∀i ≤ length ?q - 1 - 1. (nth ?q i, nth ?q (i + 1)) ∈ Y ∪ {(?v, ?
         w)} ∨ (nth ?q (i + 1), nth ?q i) ∈ Y ∪ {(?v, ?w)}" using q_el_prop
        by (metis le_neq_implies_less length_prop)
      then show ?thesis by auto
    qed
    then have "?q ≠ [] ∧ hd ?q = v ∧ last ?q = ?w ∧ (∀i < length ?q - 1. ((?q ! i
         , ?q ! (i + 1)) ∈ Y ∪ {(?v, ?w)}) ∨ (nth ?q (i + 1), nth ?q i) ∈ Y ∪ {(?v
         , ?w)})"
      using ⟨?q ≠ []⟩ hd_last_q by simp
    then show ?thesis using ⟨u = ?w⟩ ⟨length ?q ≥ 2⟩ by blast
```

The above listing deals with the first of two cases. If the path `q` goes from `v` and `?v`, then the required path is `q @ [?w]`, where `?w` is appended at the end of the path. Otherwise, the required path is `?w # q`, where `?w` is appended at the beginning. In both cases, we obtain a valid path from `v` to `?w`.

The next case when `v = ?w` follows similarly with another case split on `u`, and hence is skipped from the report.

To prove acyclic, we break down the proof into two cases - `Y = {}` and otherwise. The first case follows trivially. The second case is proven by contradiction. We first assume a cycle `pa` exists from `v` to itself. Then we have another case analysis on each element `v ∈ V2 ∪ {?w}`. If `v = ?w`, we prove the case as there's no edge that gets the cycle out of `?w` (a result called `w_el_prop` in the formalization). If `v ∈ V2`, and if `?w` doesn't lie in the cycle, then all elements of `pa` lie in `V2`, giving a cycle entirely in `V2`, disproving the fact that `V2, Y` is acyclic. The next listing shows the case when `?w` lies in the cycle at `v`.

Listing 33: *A Snippet from the proof of Acyclic Property*

```
   case False
show "False"
proof (cases "List.member pa ?w")
  case True
  have "length pa > 0" using pa_prop by simp
  then have "?w ∈ set pa" using True in_set_member pa_prop by metis
  then have "∃i < length pa. (nth pa i) = ?w" using ⟨length pa > 0⟩ in_set_conv_nth
    by metis
  then obtain i where i_prop: "(nth pa i) = ?w " " i < length pa " by auto
  have "(nth pa (length pa - 1)) = v" using pa_prop last_conv_nth by metis
  then have "(nth pa (length pa - 1)) ≠ ?w" using ⟨v ≠ ?w⟩ by simp
  then have " i ≠ (length pa - 1)" using i_prop(1) by auto
  then have "i < length pa - 1" using i_prop(2) ⟨length pa > 0⟩ by simp
  then have edge_fact: "(nth pa i, nth pa (i + 1)) ∈ Y ∪ {(⟨v⟩, ⟨?w⟩)}" using pa_prop
       by simp
  then have "(nth pa (i + 1)) ∈ V2 ∪ {?w}" using ⟨Y ∪ {(⟨v⟩, ⟨?w⟩)} ⊆ (V2 ∪ {?w}) ×
       (V2 ∪ {?w})⟩ by auto
  then show ?thesis using ⟨nth pa i = ?w⟩ w_el_prop edge_fact by blast
next
```

Hence, using the conclusions of digraph, cardinality property, acyclic and connected, we prove that `V2 ∪ {?w}` and `Y ∪ {(?v, ?w)}` form a tree containing `r` and the latter edge set is in `?F`, proving axiom 2 of greedoids. The above listings prove the claim for `(v', w) ∈ X`. We skip the proof for the case when `(w, 'v) ∈ X`, as the proof is similar to that of the previous case.

The proof of this theorem in [KV06] involves defining aborescences rooted at a fixed vertex r. This is equivalent to trees containing vertex r as per Theorem 2.4 in [KV06]. Furthermore, the proof is split into multiple cases and contradictions in order to ease formalization using Isabelle.

4.2.2 The Greedy Algorithm

To formalise the greedy algorithm, we set up a locale fixing the greedoid, an oracle that says whether a given `X` is in `F` and a list `es` that whose set is `es` and contains distinct elements. We then define the following terms - Definition 4.5, Definition 4.6 and a set when it has maximum weight.

Listing 34: *Greedy Algorithm Locale and Definitions*

```
  definition strong_exchange_property where "strong_exchange_property E F ⟺ (∀ A B x. A ∈ F ∧ B ∈ F ∧ A ⊆ B ∧ (maximal (λ B. B ∈ F) B) ∧ x ∈ E - B ∧ A ∪ {x} ∈ F ⟶ (∃ y. y ∈ B - A ∧ A ∪ {y} ∈ F ∧ (B - {y}) ∪ {x} ∈ F))"

locale greedy_algorithm = greedoid +
fixes orcl :: "'a set ⟶ bool"
fixes es
assumes orcl_correct: "∀ X. X ⊆ E ⟶ orcl X ⟺ X ∈ F"
assumes set_assum: "set es = E ∧ distinct es"

context greedy_algorithm
begin

definition valid_modular_weight_func :: "('a set ⟶ real) ⟶ bool" where " valid_modular_weight_func c = (c ({}) = 0 ∧ (∀ X l. X ⊆ E ∧ X ≠ {} ∧ l = {c (e) | e. e ∈ X} ∧ c (X) = sum (λ x. real x) l))"

definition "maximum_weight_set c X = (X ∈ F ∧ (∀ Y ∈ F. c X ≥ c Y))"
```

Then, a definition called `find_best_candidate` is formalized in which, for a given `F'` and cost function `c`, the best element (candidate) is chosen.

Listing 35: *find_best_candidate*

```
  definition "find_best_candidate c F' = foldr (λ e acc. if e ∈ F' ∨ ¬ orcl ( insert e F') then acc
  else (case acc of None ⇒ Some e |
  Some d ⇒ (if c {e} > c {d} then Some e
  else Some d))) es None"
```

The above definition iterates through all elements of list `es` and assigns the accumulator `acc` as `None`. If `e ∈ F'` or `F' ∪ e` is not in `F`, then the accumulator continues iterating through `es`. Otherwise, if `acc` is None, the best candidate remains as `e`. If `acc` has some element `d` then the costs of `e` and `d` are compared. If the latter has greater cost `acc` remains the same. Else it chooses `Some e`.

Lastly a function is defined for the greedy algorithm.

Listing 36: *The Greedy Algorithm*

```
  function (domintros) greedy_algorithm_greedoid::"'a set ⇒ ('a set ⇒ real) ⇒ 'a set" where "greedy_algorithm_greedoid F' c = (if (E = {} ∨ ¬(F' ⊆ E)) then undefined
  else (case (find_best_candidate c F') of Some e ⇒ greedy_algorithm_greedoid(F' ∪ {the (find_best_candidate c F')}) c | None ⇒ F'))"
```

The proofs that follow show that the above function is well defined and gives the same output for two inputs of the same value. The above algorithm terminates as the size of `F' ∪ {the (find_best_candidate c F')}` reduces in every iteration, for a given `F' ⊆ E` and `find_best_candidate = Some x`. We prove this as lemma `find_best_candidate_in_es` says `x` is in list `es`. Also, we obtain from lemma `find_best_candidate_notin_F'` that `x` is not in `F'`. Therefore we conclude `x ∈ E - F'` and `F' ⊂ F' ∪ {the (find_best_candidate c F') }`. Hence `E - (F' ∪ {the (find_best_candidate c F') }) ⊂ E - F'`, proving the decrease in cardinality in every iteration.

Listing 37: *Termination of the Greedy Algorithm*

```
termination greedy_algorithm_greedoid
proof (relation "measure (λ(F', c). card (E - F'))")
show "wf (measure (λ(F', c). card (E - F')))" by (rule wf_measure)
show "∀F' c x2.
     ¬ (E = ∅ ∨ ¬ F' ⊆ E) ⟹
     find_best_candidate c F' = Some x2 ⟹
     ((F' ∪ {the (find_best_candidate c F')}, c), F', c)
     ∈ measure (λ(F', c). card (E - F'))"
proof -
  fix F' c x
  show "¬ (E = ∅ ∨ ¬ F' ⊆ E) ⟹
     find_best_candidate c F' = Some x ⟹
     ((F' ∪ {the (find_best_candidate c F')}, c), F', c)
     ∈ measure (λ(F', c). card (E - F'))"
  proof -
    assume assum1: "¬ (E = ∅ ∨ ¬ F' ⊆ E)"
    show "find_best_candidate c F' = Some x ⟹
  ((F' ∪ {the (find_best_candidate c F')}, c), F', c)
  ∈ measure (λ(F', c). card (E - F'))"
    proof -
      assume assum2: "find_best_candidate c F' = Some x"
      then have "List.member es x" using find_best_candidate_in_es assum1 by auto
      then have "length es > 0" using assum1 set_assum by auto
      then have "x ∈ set es" using in_set_member ⟨List.member es x⟩ assum1 by fast
      then have "x ∈ E" using set_assum by simp
      have "x ∉ F'" using assum1 assum2 find_best_candidate_notin_F' by auto
      then have "x ∈ E - F'" using ⟨x ∈ E⟩ assum1 by simp
      then have "F' ⊂ F' ∪ {the (find_best_candidate c F')}" using ⟨x ∉ F'⟩ assum2
           by auto
      then have "E - (F' ∪ {the (find_best_candidate c F')}) ⊂ E - F'"
        by (metis Diff_insert Diff_insert_absorb ∪∅_right ∪insert_right ⟨x ∈ E - F'⟩ assum2 mk_disjoint_insert option.sel psubsetI subset_insertI)
      have "finite E" using ss_assum unfolding set_system_def by simp
      then have "finite F'" using finite_subset assum1 by auto
      then have "finite (E - F')" using ⟨finite E⟩ by blast
      then have "card (E - (F' ∪ {the (find_best_candidate c F')})) < card (E - F')"
        using ⟨E - (F' ∪ {the (find_best_candidate c F')}) ⊂ E - F'⟩
             psubset_card_mono by auto
      then show ?thesis by auto
qed
qed
qed
```

```
qed
```

Lastly, we formalize the statement that shows the correctness of the greedy algorithm for greedoids, Theorem 4.2.

Listing 38: *Greedy Algorithm Correctness*

```
lemma greedy_algorithm_correctness:
assumes assum1: "greedoid E F"
shows "(∀c. valid_modular_weight_func c ⟶ maximum_weight_set c
(greedy_algorithm_greedoid {} c)) ⟷ strong_exchange_property E F"
sorry
```

The entire formalization can be found in this link.

References

[Deo16] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Dover Books on Mathematics. Dover Publications, 2016.

[KV06] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2006.

King's College London
Department of Mathematics
Submission Cover Sheet for Coursework

The following cover sheet must be completed and submitted with any dissertation, project, coursework essay or report submitted as a part of formal assessment for degree within the Mathematics Department.

Candidate Number (this is found on your student record account):	AF27537
Module Code and Title:	7CCMMS50 Project(23~24 NSY 000001)
Title of Project/Coursework:	Introduction to Greedoids

Declaration

By submitting this assignment I agree to the following statements:

I have read and understand the King's College London Academic Honesty and Integrity Statement that I signed upon entry to this programme of study.

I declare that the content of this submission is my own work.

I understand that plagiarism is a serious examination offence, an allegation of which can result in action being taken under the College's Misconduct regulations.

Your work may be used as an example of good practice for future students to refer to. If chosen, your work will be made available either via KEATS or by paper copy. Your work will remain anonymous; neither the specific mark nor any individual feedback will be shared. Participation is entirely optional and will not affect your mark. If you consent to your submission being used in this way please add an X in the box.	X