1 Statement and Initial Thoughts

The Steinitz exchange lemma was one of the early lemmas we learned in the linear algebra module at Imperial. It was stated in the module notes (made by Dr. Charlotte Kestner), as follows:

> **Theorem 1** (Steinitz Exchange Lemma)**.** *Let V be a vector space over F. Take $X \subseteq V$ and suppose $u \in Span(X)$ but $u \notin Span(X \setminus \{v\})$ for some $v \in X$. Let $Y = (X \setminus \{v\}) \cup \{u\}$ (i.e., we exchange v for u). Then $Span(X) = Span(Y)$.*

To make the statement in Lean, the first task was to figure out a way to represent a vector space and the span of a subset. Luckily, there are standard implementation of vector spaces and spans in the mathlib documentation. Using these implementations, the formal theorem statement in Lean is

```
variable {K : Type*} [Field K] {V : Type*} [AddCommGroup V] [Module K V]
  (v u : V) (X : Set V)

theorem steinitz_exchange (hu_in : u  span K X) (hv : v  X) (hu_n_in : u  span K (X \ {v}))
    (hY : Y = ((X \ {v})  {u})): span K X = span K Y
```

One challenge I faced in the statement of the theorem was the use of the span function. In my experience before, when referring to the span of a set, we would have just said Span(X) without specifying the field. This differs to the function in Lean which required the specification of the field. Though not a large obstacle, this kind of problem was a common theme throughout the formalisation of the proof.

Some stylistic choices made were to define the variables outside the statement of the theorem and to define the hypotheses in a short but informative way, both of which are standard practices.

2 Strategy and Explanation of Formal Proof

A proof provided in the same notes is as follows:

> *Proof.* Since $u \in \text{Span}(X)$, we have $\alpha_1, \dots, \alpha_n \in F$ such that $v_1, \dots, v_n \in X$ such that
>
> $$u = \alpha_1 v_1 + \cdots + \alpha_n v_n.$$
>
> Now there is a $v \in X$ such that $u \notin \text{Span}(X \setminus \{v\})$. We may assume, WLOG, that $v = v_n$, thus $\alpha_n \neq 0$, so:
>
> $$v = v_n = \frac{1}{\alpha_n}\left(u - \alpha_1 v_1 - \cdots - \alpha_{n-1} v_{n-1}\right).$$
>
> Now if $w \in \text{Span}(Y)$, then for some $\beta_0, \beta_1, \dots, \beta_m$, we have $v_1, \dots, v_m \in X \setminus \{v\}$:
>
> $$w = \beta_0 u + \sum_{i=1}^{m} \beta_i v_i.$$
>
> Substituting $u = \alpha_1 v_1 + \cdots + \alpha_n v_n$, we get:
>
> $$w = \beta_0(\alpha_1 v_1 + \cdots + \alpha_n v_n) + \sum_{i=1}^{m} \beta_i v_i \in \text{Span}(X \setminus \{v\} \cup \{v\}) = \text{Span}(Y).$$
>
> So $\text{Span}(Y) \subseteq \text{Span}(X)$. Similarly, we have that if $w \in \text{Span}(X)$, then w is a linear combination of elements of X. Now we can replace v_n with
>
> $$\frac{1}{\alpha_n}\left(u - \alpha_1 v_1 - \cdots - \alpha_{n-1} v_{n-1}\right),$$
>
> so we can express w as a linear combination of elements of Y. So $\text{Span}(X) \subseteq \text{Span}(Y)$, thus $\text{Span}(Y) = \text{Span}(X)$. □

My first thought when reading this proof is that it constructs u and v in terms of a linear combination of vectors in X and Y and that this might be one of the challenges I might face when implementing the proof in Lean. This is because it seems like there are actually a lot of implicit steps happening in each line of the informal proof.

2.1 Iteration 1

Initially, I threw a lot of "random" tactics at the problem to see if I could arrive at a simplification of them problem and vaguely attempted to follow the reasoning of the informal proof which was

1. Find a linear combination for u using `mem_span_set'`.
2. Show that one of the vectors in the linear combination is v.
3. Show that v can be a linear combination of u and elements of $X \setminus \{v\}$.
4. Pick arbitrary $\omega \in Span(Y)$, show that it is in $Span(X)$.
5. Pick arbitrary $\omega \in Span(X)$, show that it is in $Span(Y)$.

After applying step 1 I was left with a pretty unwieldly tactic state with lots of sums and finite sequences. An example of one of the states is

```
K : Type u_1
šinst : Field K
V : Type u_2
źinst : AddCommGroup V
inst : Module K V
v u : V
X Y : Set V
hv : v  X
hY : Y = X \ {v}  {u}
n :
 : Fin n  K
b : Fin n  X
hu :  i : Fin n,  i  (b i) = u
hu_n_inX :  (x : Fin n  (X \ {v})), ň i : Fin n,  i  (x i) = u
i : Fin n
hi : (b i) = v
x : V
hx : x  X
  y  span K (X \ {v}),  z  span K {u}, y + z = x
```

Although I thought it was probably possible to work with this tactic state, I decided to stop with this approach because I found that I did not really have a direction or a plan of attack. Ofcourse there is the informal proof, but that did not serve to be the best plan for formalising in Lean. Some of the difficulties were:

1. Proving that if a linear combination is in the span of a set then the individual summands are elements of the original set. This was because we know that there is a sequence such that they are all in X but then I would also have to show that the sequence of scalars multiplied by these vectors were also in X. Though not that big of a hurdle, I thought I should try to do it in a less explicit way to make the code more readable and "Lean"-y (less explicitly constructive).
2. Using this information to construct a sequence of elements with one element being v and the others not being v and then a suitable sequence of scalars. Esentially the steps:

$$v = v_n = \frac{1}{\alpha_n}(u - \alpha_1 v_1 - \cdots - \alpha_{n-1} v_{n-1})$$

$$\text{and } w = \beta_0 u + \sum_{i=1}^{m} \beta_i v_i = \beta_0(\alpha_1 v_1 + \cdots + \alpha_n v_n) + \sum_{i=1}^{m} \beta_i v_i$$

I quickly realised that working with sums and finite sequences can get really messy so I went back to the drawing board to come up with a plan which simplifies some of these steps to avoid the above problems (at least some of it).

2.2 Iteration 2

So after my first attempt, I was more convinvced of the need for a solid plan. And because I got more familiar with some of the relevant theorems in mathlib, I could construct a plan with steps I knew could be done by existing theorems. So the plan I came up with (after lots of trial and error) and some of my thoughts about it were:

1. Split the goal of $\mathit{Span}(X) = \mathit{Span}(Y)$ into two goals $\mathit{Span}(X) \subseteq \mathit{Span}(Y)$ and $\mathit{Span}(Y) \subseteq \mathit{Span}(X)$ using `Submodule.span_eq_span` which actually lead to the pleasant simplification of the goals into $X \subseteq \mathit{Span}(Y)$ and $Y \subseteq \mathit{Span}(X)$.
2. Split $x \in X$ to $x \in X \setminus \{v\} \vee x = v$
   a. For the case $x \in X \setminus \{v\}$ we can show that $x \in \mathit{Span}(Y)$ which should be pretty easy by just showing $x \in X \setminus \{v\} \to x \in \mathit{Span}(X \setminus \{v\}) \to x \in \mathit{Span}(X \setminus \{v\} \cup \{u\}) = \mathit{Span}(Y)$.
   b. For the case $x = v$ we can show that $x \in \mathit{Span}(Y)$. This is where the challenge lies, I tried doing it within the theorem but I thought I could introduce a lemma which captures the main idea needed to prove this which was basically what the step $w = \beta_0 u + \sum_{i=1}^{m} \beta_i v_i$ in the informal proof tried to capture.
3. Split $y \in Y$ to $y \in X \setminus \{v\} \vee y = u$
   a. For the case that $y \in X \setminus \{v\}$, it should be pretty easy to show that $y \in X \setminus \{v\} \to y \in X \to y \in \mathit{Span}(X)$.
   b. For the case that $y = u$ we have that $u \in \mathit{Span} X$ by assumption.
4. Proof should be done.

This proof has a slightly different approach as the original informal proof but the main idea that "u has v encoded in it" is still maintained. The difference is that this approach should be slightly less constructive as I would not have to work so much with explicit sums representing the linear combinations in most cases. As I mentioned above, I thought I should have a helper lemma to capture the essential step in the proof.

2.2.1 Step 2b and introduction of `v_lin_comb` lemma

So to prove the case 2b, I first tried to work backwards and split down what membership in $\mathit{Span}(Y) = \mathit{Span}(X \setminus \{v\} \cup \{u\})$ actually meant. This was achieved by the rewriting $\iff$s given by the theorems `span_union`, `mem_sup` to get the goal to be `∃ y ∈ span K (X  v), ∃ z ∈ span K {u}, y + z = v`. There were other possible propositionally equal goals but I thought this was pretty direct, both to show and to use.

So I wrote down the following lemma and attempted to prove it.

```
lemma v_lin_comb (hu_in : u  span K X) (hv : v  X) (hu_n_in : u  span K (X \ {v})):
    (c : V) ( : K), c  span K (X \ {v})  u = c +   v
```

Essentially what this lemma is saying is that v is the sum of something from $\mathit{Span}(X \setminus \{v\})$ and a scalar multiple of u. To prove this lemma, the strategy was to:

1. First I showed that $u \in \mathit{Span}(X) \to u \in \mathit{Span}(X \setminus \{v\} \cup \{v\})$ which was quite the obvious statement but something I couldnt find immediately and something `by?`, `rw?` and `apply?` could not solve. So I used a have statement to introduce this as a new hypothesis to arrive at $u \in \mathit{Span}(X \setminus \{v\} \cup \{v\})$.
2. Then I showed that `∃ y ∈ span K (X \ {v}), ∃ z ∈ span K {v}, y + z = u` using `span_union` and `mem_sup`.
3. Then I extracted these witnesses using various `rcases` and substituted to show that we have $c \in V$ and $\alpha \in K : u = c + \alpha \cdot v$, closing the goal and proving the lemma.

With the lemma in place, I can use it to introduce a hypothesis that `∃ (c : V) (`α` : K), c ∈ span K (X \ {v}) ∧ u = c + `α` · v`. Now with this we can construct witnesses using `rcases` for `y` and `z` in the current goal in the theorem which is still `∃ y ∈ span K (X  v), ∃ z ∈ span K {u}, y + z = v`. These are esentially constructed as shown in the informal proof $((v =) w = \beta_0 u + \sum_{i=1}^{m} \beta_i v_i = \beta_0 u + \beta_1 y$ where $y \in \mathit{Span}(X \setminus \{v\}))$ except we dont deal with a summation of vectors but just a sum of a vector from one span and a vector from the other span. This was done by declaring new varaiables `y' := -(1/`α`)  c` and `z' := (1/`α`)  u` and hypotheses that these variables are in the right sets. Then we can close case 2b using a short straightforward calc

```
calc
  y' + z' = -(1/)  c + (1/)  u := by dsimp [y', z']
  _ = -(1/)  c + (1/)  (c +   v) := by rw [hu_eq_cav] -- hypothesis from lemma
  _ = -(1/)  c + (1/)  c + (1/)    v := by simp
  _ = ź    v := by simp
  _ = v := by rw [smul_smul, mul_comm, Field.mul_inv_cancel, one_smul]
```

I unexpectadly ran into a problem here, the final step of the proof.

2.2.2 Plot Twist

Turns out that when I was writing my lemma, it was originally

```
lemma v_lin_comb (hu_in : u  span K X) (hv : v  X) (hu_n_in : u  span K (X \ {v})):
     (c : V) ( : K), c  span K (X \ {v})  u = c +   v
```

and the linter raised a warning that I hadn't used the hypothesis `(hu_n_in :  u` $\notin$ `span K (X \ {v}))`. So without thinking much about it I removed it and thought that I had generalised the lemma. The problem was that in order to get α^{-1} α `v = v`, we need that $\alpha \neq 0$. So I went back and amended the lemma to finally be

```
lemma v_lin_comb (hu_in : u  span K X) (hv : v  X) (hu_n_in : u  span K (X \ {v})):
   (c : V) ( : K),   0  c  span K (X \ {v})  u = c +   v
```

so that we could have the hypothesis that $\alpha \neq 0$. To prove this in the lemma, I used a simple contradiction argument assuming that $\alpha = 0$ which would mean that $u \in Span(X \setminus \{v\})$ which contradicts `(hu_n_in :  u` $\notin$ `span K (X \ {v}))`, proving $\alpha \neq 0$. With this short ammendment, I could close the goal in calc using exact hypothesis that $\alpha \neq 0$. Case 2b was done.

2.2.3 Other cases

The other cases were pretty straightforward and followed exactly the line of reasoning in the plan. With all cases covered the formal proof of the Steinitz Exchange Lemma was done.

3 Extension : Replacement Theorem and Informal Proof

As proving the Steinitz Exchange Lemma took fewer lines than I thought after I made the steps more direct, I thought I could add a corollary of the lemma. One neat applications which follows from the Steinitz Exchange lemma is a replacement theorem which says

> **Theorem 2** (A Replacement Theorem). *Let V be a vector space over a field F and $X \subseteq V$. Then $\exists\, Y \subseteq V : X \cap Y = \emptyset \wedge Span(X) = Span(Y)$.*

Essentially, what this theorem is saying is that given a subset of a vector space, we can construct another subset of new elements which have the same span as the original set. I am not sure if this result is actually called the replacement theorem but we can call it that in this paper. However, while implementing this and trying to prove it I realised that it was false and that adding assumptions of infinite field and finite set X should fix the problem. So I adjusted it to the following:

> **Theorem 3** (A Replacement Theorem). *Let V be a vector space over an infinite field F and finite $X \subseteq V$. Then $\exists\, Y \subseteq V : X \cap Y = \emptyset \wedge Span(X) = Span(Y)$.*

(There are still errors to this statement which will be addressed later.) There are other versions of the replacement theorem online but they mainly assume linear independence. Therefore, I thought I should probably come up with a proof on my own where we can use the Steinitz Exchange Lemma. As I learnt in the formalisation before, I need a game plan. So I came up with the following informal proof sketch:

1. Induction on the size of the set X. The proof of the base case is trivial because if $X = \emptyset$, then we have that $Y = \emptyset$ satisfies both $Span(X) = Span(Y)$ and $Y \cap X = \emptyset$.
2. The tricky part is the induction step in which I have to assume the hypothesis holds for $n \leq k$ and show that it holds for $n = k + 1$, ie. For $n = k + 1 = |X|$ we have that for $v \in X, |X \setminus \{v\}| = k$ so by the induction hypothesis we have that $\exists\, Y' \subseteq V : Span(X \setminus \{v\}) = Span(Y') \wedge (X \setminus v) \cap Y = \emptyset$. First we know that $Span(Y') = Span(X \setminus \{v\}) \subseteq Span(X)$ so we split here into two cases: a) $Span(X) \subseteq Span(Y')$ and b) $Span(X) \nsubseteq Span(Y')$ and show we can find a suitable Y to prove the induction.
   (a) For the first case, this would automatically mean that $Span(X) = Span(Y')$. If $v \notin Y'$ then $X \cap Y' = (X \setminus \{v\} \cup \{v\}) \cap Y' = (X \setminus \{v\} \cap Y') \cup (\{v\} \cap Y') = \emptyset$ and we showed that $Y = Y'$ works. However, if $v \in Y'$, then we need to find another vector which is not in X to replace the $v \in Y$ to create a suitable Y. Because X is finite and F is infinite $\exists\, b = \alpha \cdot v : b \notin X$. Then setting $Y = Y' \setminus \{v\} \cup \{b\}$ is suitable because we have that $Span(Y) = Span(Y' \setminus \{v\} \cup \{b\}) = Span(Y') = Span(X)$ and we can show $X \cap Y = (X \setminus \{v\} \cup \{v\}) \cap (Y' \setminus \{v\} \cup \{b\}) = \emptyset$.

(b) For the second case when $Span(X) \nsubseteq Span(Y)$, then that means $\exists\ u \in Span(X) : u \notin Span(Y) = Span(X \setminus \{v\})$, a familiar result. We can then use the steinitz exchange lemma to show that $Span(X) = Span(X \setminus \{v\} \cup \{u\})$. We can then find a scalar multiple of u such that $\alpha \cdot u \notin X$. Then we have that $Y = Y' \cup \{\alpha \cdot u\}$ is suitable as $Span(X) = Span(X \setminus \{v\} \cup \{\alpha \cdot u\}) = Span(Y)$ and we can show $X \cap (Y \cup \{\alpha \cdot u\}) = (X \setminus \{v\} \cup \{v\}) \cap (Y \cup \{\alpha \cdot u\}) = (\{v\} \cap Y) \cup (X \setminus \{v\} \cap \{\alpha \cdot u\}) = \emptyset$.

3. Because we've shown the base case and the induction case, the proof is done.

Some parts of the proof which I thought might be challenging are the little equalities and lemmas we know are true but lean might not. For example, showing that because the field is infinite and X is finite, we can find a scalar multiple of any element of X which is not in X (not true for 0). Another lemma would be that if we replace an element of a set with a scalar multiple (not equal to 0) of the element then the span is the same (which is not exactly given by the steinitz exchange lemma).

3.1 Implementation

The statement and proof of the theorem is given by:

```
theorem replacement_thm [Infinite K] (hX_fin : X.Finite) :
     Y : Set V, Y.Finite  span K X = span K Y  X  Y =  := by
  -- do induction on the finite set X
  have h_base :  Y : Set V, Y.Finite  span K  = span K Y    Y =  := , by simp

  induction X, hX_fin using Finite.induction_on with
  | empty =>
    exact h_base
  | @insert a S ha hs ih =>
    exact induction_hyp a S ha hs ih
```

The induction step was done using `Finite.induction_on`. Furthermore, a few helper lemmas were created which are stated as:

```
lemma exists_scal_mul [Infinite K] (hX_fin : X.Finite) (x : V) :
      : K,   1  (  x  X)

lemma span_replaced_scal_mul [Infinite K] (Y : Set V) (hY_fin : Y.Finite) ( : K) (v : V) (hv : v  X)
 :
    span K X = span K (X \ {v}  {  v})

lemma induction_step [Infinite K] (v : V) (X : Set V) (hv_n_in : v  X) (hX_fin : X.Finite)
    (hk :  Y : Set V, Y.Finite  span K X = span K Y  X  Y = ) :
    ( Y : Set V, Y.Finite  span K (insert v X) = span K Y  (insert v X)  Y = )
```

The first and second being crucial in finding valid replacements and the third giving us the induction step where the bulk of the work was done. This turned out to be much longer than I thought it would be because showing steps like $X \cap (Y \cup \{\alpha \cdot u\}) = (X \setminus \{v\} \cup \{v\}) \cap (Y \cup \{\alpha \cdot u\}) = (\{v\} \cap Y) \cup (X \setminus \{v\} \cap \{\alpha \cdot u\}) = \emptyset$ turn out to be more difficult than I forsaw even though they seem pretty straighforward and the proof quickly got out of hand.

The second lemma was sorried because I could not figure out in time how to prove it, let alone formalise it in lean. The argument would have to do with X being finite and the field being infinite and so by some pigeon hole idea, there would be a way to construct a vector not equal to the original one which that is not in X. This is close to the Steinitz lemma because we are exchanging something which we dont know is linearly independent from the rest of the set. Next for `span_replaced_scal_mul`, this seems pretty straightforward so I focused my time on proving more significant parts of the main theorem.

A little note on notation, X here refers to `insert v X` in the code and $X \setminus \{v\}$ refers to `X` in the code.

3.2 The `induction_step` Lemma

As can be from the short proof of the theorem, the bulk of the work in the extension lied elsewhere. In the induction step, I had to show that given finite $X' \subseteq V$ and a Y' which satisfies the theorem, we can find another Y which satisfies the conditions for $X =$ `insert v X` and `v` $\notin$ `X`. First I broke apart the assumption using `rcases` and subsequently split it into two cases by showing that either $Span(X) \subseteq Span(Y')$ or $Span(X) \nsubseteq Span(Y')$ using the law of excluded middle. Originally, I had the proof for both cases under the `induction_step` lemma but I decided that maybe I should split this up because it was running quite long.

3.2.1 Case 1 - `induction_step_left` lemma: $Span(X) \subseteq Span(Y')$

For the first case, the statement of the lemma is

```
lemma induction_step_left [Infinite K] (hv_n_in : v  X) (hX_fin : X.Finite) (Y’ : Set V)
    (hY’_fin : Y’.Finite) (h_span_eq : span K X = span K Y’) (h_inter_empty : X  Y’ = )
    (h : span K (insert v X)  span K Y’) :
    ( Y, Y.Finite  span K (insert v X) = span K Y  insert v X  Y = )
```

As $Span(X) \subseteq Span(Y')$, this would mean that $Span(X) = Span(Y')$ so I had to prove that as a hypothesis which I sorried. Next, the strategy was to first deal with the possibility that v was in Y which would mean that $X \cap Y' \neq \emptyset$. So I applied the `exists_scal_mul` lemma to get a suitable replacement and replace $v \in Y'$ to get Y. Next, I had to show that this Y is finite using `Finite.union (Finite.diff hY_fin) (finite_singleton u)`. Next because it was a scalar replacement and we have $Span(X) = Span(Y')$ and we can get $Span(Y') = Span(Y)$ using our `span_replace_scal_mul` lemma, we can combine it with a transitivity argument to show that $Span(X) = Span(Y)$. Next I had to show that the intersection with our new X and Y is empty which turned out to be very tedious and long `rws`. So in conclusion, this section has one main sorry, the first showing that $Span(X) = Span(Y')$.

3.2.2 Case 2 - `induction_step_right` lemma: $Span(X) \not\subseteq Span(Y')$

The main point of this section is to show that I can use the Steinitz Exchange Lemma from above in proof of the replacement theorem. The statement of the lemma for this step is

```
lemma induction_step_right [Infinite K] (hv_n_in : v  X) (hX_fin : X.Finite) (Y’ : Set V)
    (hY’_fin : Y’.Finite) (h_span_eq : span K X = span K Y’) (h_inter_empty : X  Y’ = )
    (h : ňspan K (insert v X)  span K Y’) :
     Y, Y.Finite  span K (insert v X) = span K Y  insert v X  Y =
```

It starts of by setting the stage to use the Steinitz Exchange lemma where I had to do the following to go from the hypothesis that ¬`span K (insert v X)` $\leq$ `span K Y’` to the hypotheses `u :  V, hu_nin :  u` $\notin$ `span K X, hu_in :  u` $\in$ `span K (insert v X)` which is where I first began to question whether my Steinitz Lemma was in a helpful form. Another hurdle was understanding what $\leq$ meant in the context of spans and how I could go from one to the other using `span_le` and subsequently promoting it to a subset of V using `simp`.

```
rw [span_le, subset_def,  h_span_eq] at h
push_neg at h
rcases h with u, hu_in, hu_nin
simp at hu_nin
apply @subset_span K at hu_in
simp at hu_in
```

The next annoying thing was that I had at this point I had kind of shown `u` $\in$ `span K (insert v X)` and `u` $\notin$ `span K X` which is not exactly what I need because the arguments of steinitz exchange lemma are a proof of `u` $\in$ `Span insert v X` and `u` $\notin$ `Span (insert v X` $\setminus$ `{v})` so first i prepare an intermediate set Z = insert v X and the right hypotheses for Z using the out assumptions about u. This entailed a few more tedious steps again bolstering my view that the steinitz lemma was in the best form for application here.

```
let Z := insert v X
let W := (Z \ {v}  {u})
have hW : W = (Z \ {v}  {u}) := rfl
have hu_inZ : u  span K Z := hu_in
have hu_ninZv : u  span K (Z \ {v}) := by sorry
have hZ_eq : Z \ {v} = X := by sorry
have hv_in : v  Z := mem_insert v X

have h_span_XeqY : span K Z = span K (Z \ {v}  {u}) := by
  exact @steinitz_exchange K _ V _ _ v u _ _ hu_inZ hv_in hu_ninZv hW
```

After applying the Exchange lemma I performed a usual swap with using the `exists_scal_mul` lemma and then built `Y := (Z` $\setminus$ `{v}` $\cup$ `{`α $\cdot$ `u})`. Next what was left was to show the three conditions of finiteness, equality of span and empty intersection, the first of which was straightforward and done with the `Finite.union` function which gives a proof that a union is finite given proofs that the sets being union-ed are finite. The other two goals were sorried because I ran out of time.

3.3 Plot Twist 2 - More work to be done

One case I hadn't mentioned so far is the case that X is the singleton 0 vector. I had not considered this case so far and it would break my theorem. In this case the span of X is 0 and this cannot be replaced by any other different element. Where this popped up was when I was trying to get rid of the sorry for one particular have statement.

```
    have h_vu_disjoint : Disjoint ({v} : Set V) {  v}
```

This hypothesis was crucial because it ensured that we could have an empty intersection between the original set X and the new set Y. To amend and account for this error we need to further place a restriction on the theorem and amend the proofs. This was not done due to the lack of time. However, I can change the assumption to none of the elements in X being the 0 vector which would be an even weaker statement but simpler proof. So I changed it and amended some of the lemmas and theorems to have this assumption and finally the theorem looked like this

```
  theorem replacement_thm [Infinite K] (hX_fin : X.Finite) (h0notinX : 0  X) :
       Y : Set V, Y.Finite  span K X = span K Y  X  Y =  := by
    -- do induction on the finite set X
    have h_base :  Y : Set V, Y.Finite  span K  = span K Y   Y =  := , by simp

    induction X, hX_fin using Finite.induction_on with
    | empty =>
      exact h_base
    | @insert a S ha hs ih =>
      have h0notinS : 0  S := by
        intro h
        apply h0notinX
        rw [mem_insert_iff]
        right
        exact h
      have ha_ne_0 : a  0 := by
        by_contra ha_eq_0
        rw [ha_eq_0, insert_eq, mem_union] at h0notinX
        tauto
      exact induction_step a S ha hs (ih h0notinS) ha_ne_0
```

I had also adjusted the lemmas and included this hypothesis which is provided by the main theorem by inserting an extra hypothesis that `v` $\neq$ `0`.

4 Reflections

4.1 Concrete changes for next time

One thing I was made aware of throughout this whole process was the need to formulate functions in an optimal way. There are many equivalent formulations of theorems but some prove to be much more useful than the other. Despite having done the steinitz exchange lemma as stated in the 1-st year linear algebra notes, maybe I should have stated it as:

```
    theorem steinitz_exchange (hu_in : u  span K (insert v X)) (hu_n_in : u  span K X) : span K insert v
    X = span K insert u X
```

This statement feels like it could be more useful and allow people to operate in the more lean version of definitions and avoid having to use `insert_eq` to constantly go from `insert v X` to `X ∪ {v}`.

Next, I think it would probably be a good idea to organise the code into different files and declare variables to reducde the number of hypotheses and variables in the names of the theorems and lemmas. Furthermore, I could have done a better job of using tactics efficiently and making use of functions and term mode.

Additionally, as I could not complete the proof I am fully aware that there might be holes which might pop up because many hypotheses have sorried proofs. This might lead to me having to introduce further assumptions and weaken the replacement theorem like I initially had to and later had to with the restriction that X cannot be the singleton of the 0 vector. I had underestimated the length of this proof and could not complete it in time. However, I am sure I can finalise some version of it in the future.

4.2 General reflection

One of the things that stuck out the most was the need to explain to Lean every single step. Despite it taking much more time that I thought it would take, I did largely appreciate this because it brought me back down a few levels to the nuts and bolts of mathematics, which made me feel like I really knew the proofs of theorems inside out. Another related point of reflection was that sometimes the informal proof is not the best proof for lean because while little steps might be obvious in informal proofs, you really do have to go through each little step. So what it feels like to me is that sometimes proofs with few big steps might be easier to do in Lean than proofs with many little steps.

Additionally, at the start, I struggled with finding the right tactics to do what I wanted to achieve. I intially just worked with a few simple ones and later on when I had to do more complicated tasks, I searched for ways to combine tactics or tactics which would help simplify the proof. One such tactc was `obtain`, which I did not initially use but proved useful when I had to extract witnesses. Similarly, I also spent a lot of time looking through Mathlib using Leansearch to find the right theorems and this was not the most easy task because even though you know exactly the result you want to find, sometimes the implementation is in a different format or sometimes it is named cryptically.

Overall, I really liked using Lean because it was a good challenge and it was a really humbling experience taking on these theorems we learned in the first year and finding it really difficult. I appreciated the support from my class mates and teachers and that sometimes when I got stuck they would help me figure out what was going wrong. Some specific people I would like to thank are Bhavik, David, Kevin Buzzard, Dirichlet @thedirichlet on Xena Discord and Ben @undefined2338 for their help in fixing specific problems which are noted in the lean file.