Treball Fi de Grau - Curs 2023/2024

Verificació formal de formes equivalents de l'axioma d'elecció

Autor: **Vicent Pons Llopis**

Tutor: Enric Cosme Llópez

Facultat de Matemàtiques Grau en Matemàtiques

Dedicatòria

Dedique aquest treball final de grau a les persones que han sigut la meua font d'inspiració i suport al llarg d'aquest viatge acadèmic. En primer lloc, al meu estimat mestre Enric, que fa dos anys em va introduïr en el fascinant món de Lean i ha sigut una influència inestimable en el meu creixement acadèmic. Als meus pares, a la meua germana i amics, vos estic profundament agraït pel vostre recolzament incondicional al llarg d'aquests anys. Sense la vostra confiança i suport, aquest resultat no hagués sigut possible. I finalment, vull dedicar aquest treball a la meua estimada àvia, que, amb la seua passió pels escacs i les matemàtiques em va transmetre la fascinació per aquesta ciència. A tots vosaltres la meua més profunda gratitud.

Índex

CAPíTOL 1

Introducció

La matemàtica està intrínsecament lligada a la recerca de veritat i rigor. A través dels segles, els matemàtics han perseguit la comprensió profunda de les estructures i les relacions que formen la base de les seues teories. En aquest context, el Treball de Fi de Grau que presentem és una exploració rigorosa en diverses de les equivalències de l'axioma d'elecció i la seua formalització en el sistema de demostració assistida per ordinador, Lean.

L'axioma d'elecció és un enunciat matemàtic que, tot i la seua aparent simplicitat, ha desconcertat i inspirat matemàtics durant dècades. La seua relació amb altres enunciats en la teoria de conjunts és complexa i fascinant. En aquest treball aprofundirem en diverses equivalències que obtenim a partir d'aquesta proposició.

A més de les demostracions en paper, aquest treball farà ús d'un mitjà tecnològic poderós: el sistema de demostració assistida de codi obert Lean. Aquesta integració de tecnologia i matemàtica ressalta la manera en què les noves eines poden redefinir la manera en què els matemàtics avancem en el coneixement.

A continuació, presentarem breument el contingut de cada capítol per a proporcionar una visió general de l'estructura del treball i el que es pot esperar.

En el primer capítol, explorarem la teoria de conjunts, un dels fonaments de les matemàtiques, i estudiarem els seus axiomes fonamentals que estableixen les regles bàsiques per a la construcció i manipulació de conjunts.

El segon capítol tractarà sobre diverses demostracions teòriques que establixen les equivalències amb l'axioma d'elecció. Mitjançant aquestes demostracions rigoroses, mostrarem com l'Axioma d'Elecció és equivalent a altres afirmacions, establint una base sòlida per a la seua comprensió i aplicació.

El tercer capítol s'introdueixen els conceptes fonamentals de la ferramenta de demostració assistida Lean, i una breu introducció a la teoria de tipus. A més, seguirem les equivalències de l'axioma d'elecció fetes a Lean, conectant, els conceptes teòrics amb la formalització

en Lean. Aquesta demostració completa la podem trobar a la següent referència [**PL23**].

Finalment, el quart capítol serveix com a conclusió del treball. Recapitularem els punts claus dels capítols anteriors i oferirem una visió global del treball realitzat.

Amb aquesta introducció, donem inici a una investigació sobre les equivalències amb l'axioma d'elecció i la seua formalització en Lean.

CAPíTOL 2

Teoria de Conjunts: Axiomes i Fonaments

En aquest capítol, explorarem la teoria de conjunts, un dels fonaments de les matemàtiques, i estudiarem els seus axiomes que estableixen les regles bàsiques per a la construcció i manipulació de conjunts. Més endavant estudiarem en profunditat l'axioma central del treball, l'axioma d'elecció.

Cal destacar que, al llarg d'aquest treball, els conceptes que exposarem sobre teoria de conjunts es basen, en la seua gran majoria, en els que s'han descrit a [**CV10**].

1. Història de la Teoria Axiomàtica de Conjunts

La teoria axiomàtica de conjunts té les seues arrels en els treballs pioners de matemàtics com Georg Cantor, Richard Dedekind i Giuseppe Peano a finals del segle XIX. Els intents de formalitzar les matemàtiques i d'establir fonaments sòlids per a aquesta disciplina van portar a la creació de la teoria axiomàtica de conjunts com una estructura formal per a les matemàtiques.

La Paradoxa de Russell. L'any 1901, el filòsof i lògic Bertrand Russell va presentar una paradoxa que va posar en qüestió els fonaments de la teoria de conjunts. Aquesta paradoxa, coneguda com la "Paradoxa de Russell", es basava en la següent qüestió:

Considerem el conjunt R que consisteix en tots els conjunts que no es contenen a si mateixos com a elements. La pregunta era: $R \in R$ o $R \notin R$? Si $R \in R$, aleshores R ha de ser un element d'ell mateix i, per tant, no pot estar en R. Però si $R \notin R$, aleshores compleix la condició per a estar en R.

Aquesta contradicció va posar en rellevància una falta de rigor en els fonaments de la teoria de conjunts.

Solució de Zermelo. Pe ar resoldre la Paradoxa de Russell i altres paradoxes, el matemàtic Ernst Zermelo va formular una sèrie d'axiomes que van establir les bases per a la teoria de conjunts moderna. Aquests axiomes es coneixen com els "axiomes de Zermelo-Fraenkel" (ZF) i, juntament amb l'axioma de l'elecció, formen la base de la teoria de conjunts àmpliament acceptada a dia de hui (ZFC).

Entre altres, els axiomes de ZF aborden les qüestions de la formació de conjunts, l'existència d'un conjunt buit, la comprensió, la unió, la intersecció i altres operacions bàsiques sobre els conjunts. Aquests axiomes van proporcionar una estructura coherent i lliure de paradoxes per a la teoria de conjunts i van allunyar les preocupacions inicials sobre les incoherències de la teoria.

2. Teoria Axiomàtica de Conjunts

La teoria de conjunts es basa en una sèrie d'axiomes, principis fonamentals que estableixen les regles bàsiques que tots els conjunts han de satisfer. Aquesta aproximació axiomàtica és crucial per a garantir la coherència i l'absència de paradoxes dins de la teoria de conjunts. A continuació, presentem els axiomes que defineixen aquest marc teòric:

Axioma d'extensionalitat. L'axioma d'extensionalitat estableix que dos conjunts són iguals si, i només si, tenen els mateixos elements. Formalment, si X i Y són dos conjunts, llavors

$$(X = Y) \longleftrightarrow \forall x(x \in X \leftrightarrow x \in Y). \tag{1}$$

Axioma del conjunt buit. L'axioma del conjunt buit estableix que existeix un conjunt buit, és a dir, un conjunt que no conté cap element. Aquest axioma és essencial per a iniciar la construcció de tots els altres conjunts. La seua definició formal seria la següent:

$$\exists X, \forall x(x \notin X). \tag{2}$$

A partir d'aquest axioma podem demostrar que aquest conjunt és únic:

Lema 2.1. *El conjunt buit és únic.*

Demostració. Considerem X i Y dos conjunts buits. Volem demostrar que són iguals. Donat un conjunt x, si x és un element de X, com X és buit, obtenim una contradicció. Per tant, podem confirmar

$$x \in X \longrightarrow x \in Y. \tag{3}$$

De forma anàlega podem demostrar la implicació contrària i per tant, obtindre, per l'axioma d'extensionalitat (1), el resultat que volíem demostrar. □

Com només existeix un únic conjunt buit, el denotarem per $\emptyset$.

Axioma del conjunt parell no ordenat. L'axioma del conjunt parell no ordenat ens permet construir un conjunt que conté dos elements determinats sense especificar-ne l'ordre. Això s'utilitza per a definir parelles no ordenades com $\{a, b\}$. Formalment s'especifica de la següent manera:

$$\forall x, y, \exists X, \forall w \in X(w = x \vee w = y). \tag{4}$$

Axioma del conjunt unió. L'axioma del conjunt unió ens permet formar un conjunt que conté tots els elements d'altres conjunts. Si tenim un conjunt de conjunts $\mathcal{X}$, podem construir la unió de tots els elements de $\mathcal{X}$ com $\bigcup \mathcal{X}$.

$$\forall \mathcal{X}, \exists \mathcal{Y}, \forall w(w \in \mathcal{Y} \leftrightarrow \exists X(w \in X \wedge X \in \mathcal{X})). \tag{5}$$

Definició 2.2 (Subconjunt). *Donats dos conjunts X i Y, direm que X és subconjunt de Y si,*

$$\forall x, (x \in X \longrightarrow x \in Y). \tag{6}$$

Aquesta relació la denotem per $X \subseteq Y$.

Axioma del conjunt potència. L'axioma del conjunt potència ens permet construir el conjunt de totes les subcol·leccions d'un conjunt X.

$$\forall X, \exists \mathcal{X}, \forall Y(Y \in \mathcal{X} \leftrightarrow Y \subseteq X). \tag{7}$$

Es pot demostrar que, per a un conjunt X el seu conjunt potència és únic, el denotarem per $\mathcal{P}(X)$.

Esquema axiomàtic de separació. L'esquema axiomàtic de separació és un dels pilars fonamentals de la teoria de conjunts i permet la construcció de nous conjunts mitjançant la selecció d'elements d'un conjunt existent que compleixen una condició específica. Aquest esquema s'ha formulat de diferents maneres al llarg del temps, amb aportacions importants de Zermelo, Fraenkel i Skolem.

Zermelo va ser el primer en formular l'esquema de separació, utilitzant el concepte informal de "definite Eigenschaft" (propietat ben definida) per a establir les bases de la construcció de nous conjunts.

No obstant això, Fraenkel i Skolem van criticar l'enfocament de Zermelo i van proposar una reformulació més precisa utilitzant la lògica de predicats de primer ordre amb igualtat. Aquest enfocament va ser més restrictiu i va assegurar que les condicions imposades per l'esquema foren clarament definides i precises.

La formulació essencial de l'esquema axiomàtic de separació estableix que, donat els conjunts X, $t_0, \ldots, t_{n-1}$ i una condició $\varphi(x, t_{[n]})$ ben

definida en el llenguatge de la teoria de conjunts, existeix un nou conjunt Y que consisteix en tots els elements de X que satisfan la condició $\varphi(x, t_{[n]})$. Això es representa com:

$$Y = \{x \in X : \varphi(x, t_{[n]})\}.$$

On x és una variable que representa els elements de X, i , $t_0, \ldots, t_{n-1}$ la resta de conjunts distints de Y que s'utilitzen com a variables a $\varphi(x, t_{[n]})$, que és la condició específica que determina quins elements es seleccionen per formar el nou conjunt Y.

L'ús d'aquest esquema és fonamental per a la construcció de subconjunts i la definició precisa de conjunts amb propietats específiques. Aquesta eina és crucial per a la formalització i la construcció de les estructures matemàtiques dins de la teoria de conjunts.

En les seccions següents d'aquest treball, explorarem com s'utilitza la teoria de conjunts com a fonament per a altres àrees de les matemàtiques i com aquesta teoria s'implementa en sistemes com Lean per a la verificació formal.

3. Definicions bàsiques de teoria de conjunts

En aquesta secció, establim definicions fonamentals que són essencials per comprendre la teoria dels conjunts.

Definició 2.3 (Unió de conjunts). *Per a definir la unió de dos conjunts X i Y, utilitzem l'axioma del conjunt parell no ordenat per a crear el conjunt $\{X, Y\}$. Llavors, definim la unió entre X i Y com $\bigcup\{X, Y\}$, fent servir l'axioma de la unió. Això resulta en un conjunt que conté tots els elements de tots dos conjunts. Denotem aquesta unió com a $X \cup Y$.*

Definició 2.4 (Intersecció de conjunts). *La intersecció entre els conjunts X i Y es defineix com el conjunt d'elements que pertanyen tant a X com a Y i s'expressa com a:*

$$X \cap Y = \{x \in X : x \in Y\}. \tag{8}$$

Definició 2.5 (Diferència de conjunts). *La diferència entre els conjunts X i Y es defineix com el conjunt d'elements que pertanyen a X, però no a Y i s'expressa com a:*

$$X \setminus Y = \{x \in X : x \notin Y\}. \tag{9}$$

Definició 2.6 (Parell ordenat de Kuratowski). *Definim el parell ordenat format pels conjunts X i Y com:*

$$(X, Y) = \{\{X\}, \{X, Y\}\}. \tag{10}$$

Aquesta no és la única manera que hi ha de definir parells ordenats, però és una de les més àmpliament acceptades i que utilitzarem al llarg d'aquest treball, ja que és la utilitzada en Lean.

DEFINICIÓ 2.7 (Producte cartesià de conjunts). *El producte cartesià entre dos conjunts X i Y es defineix com el conjunt de tots els parells ordenats (x, y), on x pertany a X i y pertany a Y, i s'expressa com a:*

$$X \times Y = \{(x, y) \in \mathcal{P}(\mathcal{P}(X \cup Y)) : x \in X \wedge y \in Y\}. \tag{11}$$

DEFINICIÓ 2.8 (Funcions en teoria de conjunts). *Direm que un conjunt f és una funció amb domini X i codomini Y, si i només si, $f \subseteq X \times Y$ i donat x un element de X, existeix un únic y que pertany a Y tal que $(x, y) \in f$. Denotem per* $\mathrm{Fun}(X, Y)$ *al conjunt de totes les funcions que tenen com a domini X i com a codomini Y.*

CAPíTOL 3

Equivalències de l'Axioma d'Elecció

En aquest capítol, explorarem diverses demostracions teòriques que establixen les equivalències amb l'Axioma d'Elecció. L'Axioma d'Elecció, malgrat la seua aparent simplicitat, té una relació complexa amb altres enunciats i conceptes en la teoria de conjunts. Demostrarem que aquesta presentació és equivalent a altres enunciats, establint una base sòlida per a la seua comprensió i aplicació.

És important destacar que la versió que anem a presentar de l'Axioma d'Elecció que va formular Zermelo no és l'axioma com va ser desenvolupat originalment, sinó una forma equivalent. L'utilitzarem com alternativa per a les nostres demostracions i en aquest mateix capítol, demostrarem la seua equivalència en l'Axioma d'Elecció original.

1. Axioma d'Elecció de Zermelo

Abans de procedir a la definició de l'Axioma d'Elecció, és crucial definir diversos conceptes que ens permetran comprendre plenament aquest enunciat.

DEFINICIÓ 3.1 (Disjunt). *Direm que dos conjunts X, Y són disjunts si la intersecció és buida i ho escriurem com $X \perp Y$. Siga $\mathcal{X}$ un conjunt de conjunts, direm que és disjunt, si agafant dos elements qualsevol del conjunt, aquestos són disjunts. Ho denotarem com* $\mathrm{Disj}(\mathcal{X})$.

AXIOMA 3.2 (Axioma d'Elecció (AC)). *Siga $\mathcal{X}$ un conjunt de conjunts disjunt que, a més, no conté el conjunt buit, aleshores existeix una funció*

$$F : \mathcal{X} \longrightarrow \bigcup \mathcal{X}$$

tal que, per a cada X en $\mathcal{X}$ es compleix que $F(X)$ és un element de X. Aquestes funcions s'anomenen funcions d'elecció.

NOTA 3.3. *Com s'ha comentat a l'inici del capítol, Zermelo va definir l'axioma (AC) sense afegir la condició de que el conjunt $\mathcal{X}$ siga disjunt, però aquest és equivalent a l'axioma que hem definit. A continuació demostrarem aquesta equivalència.*

TEOREMA 3.4. *L'Axioma d'Elecció no depén de que el conjunt $\mathcal{X}$ siga disjunt.*

DEMOSTRACIÓ. Per a resoldre aquesta proposició només cal demostrar que si l'axioma es compleix per a tot conjunt disjunt, aleshores es compleix per a tot conjunt de conjunts, independent que siga o no disjunt.

Considerem $\mathcal{X}$ un conjunt de conjunts que no conté al conjunt buit. Per a cada conjunt X en $\mathcal{X}$, definim un conjunt etiquetat com $X \times \{X\}$. D'aquesta manera, per a cada parella de conjunts X i Y en $\mathcal{X}$, els conjunts $X \times \{X\}$ i $Y \times \{Y\}$ són disjunts dos a dos. Així doncs, podem definir un conjunt que continga tots els conjunts de $\mathcal{X}$ amb etiqueta, i l'anomenem $\coprod \mathcal{X}$. Així,

$$\coprod \mathcal{X} = \bigcup_{X \in \mathcal{X}} X \times \{X\}.$$

Ara podem aplicar l'Axioma d'Elecció (AC) a aquest conjunt, ja que sabem que és un conjunt disjunt i, com $\mathcal{X}$ no conté al conjunt buit, aquest conjunt tampoc el conté.

Per tant, obtenim l'existència d'una funció d'elecció

$$H : \coprod \mathcal{X} \longrightarrow \bigcup \left(\coprod \mathcal{X} \right) \tag{12}$$

Aquesta funció fa correspondre cada $X \times \{X\}$ de $\coprod \mathcal{X}$ amb un element (x, X) que pertany a $X \times \{X\}$.

Aleshores, si definim les aplicacions següents:

$$\begin{array}{rccc} \text{in}: & \mathcal{X} & \longrightarrow & \coprod \mathcal{X} \\ & X & \longmapsto & X \times \{X\} \end{array} \tag{13}$$

i la funció projecció de la primera component, π_1, obtenim:

$$F = \pi_1 \circ H \circ \text{in}, \quad \text{on} \quad F : \mathcal{X} \longrightarrow \bigcup \mathcal{X}.$$

D'aquesta manera, a cada X de $\mathcal{X}$ li correspon un element x que, per la definició de H, pertany a X. Per tant, F és una funció d'elecció i amb açò ja queda demostrada l'equivalència. □

2. Funció sobrejectiva i inversa per la dreta

En aquesta secció, explorarem una equivalència que relaciona l'Axioma d'Elecció amb que tota funció sobrejectiva té inversa per la dreta. Aquest resultat destaca la rellevància de l'Axioma d'Elecció dins de la teoria de conjunts i com es relaciona amb conceptes clau de les matemàtiques.

Malgrat que aquesta proposició pot semblar intuïtiva a primera vista, el fet que siga equivalent a l'Axioma d'Elecció li dona una rellevància significativa. Això ens mostra com conceptes que aparentment són simples poden estar intrínsecament lligats a l'estructura profunda de la teoria de conjunts.

Al llarg d'aquesta secció, farem referència a X i Y com a conjunts genèrics.

DEFINICIÓ 3.5 (Funció sobrejectiva). *Donada una funció $f : X \longrightarrow Y$, direm que és* sobrejectiva *si per a tot element y de Y existeix almenys un element x de X tal que $f(x) = y$. És a dir, tot element de Y té una preimatge no buida en X.*

DEFINICIÓ 3.6 (Inversa per la dreta). *Donada una funció $f : X \longrightarrow Y$, direm que $g : Y \longrightarrow X$ és una* inversa per la dreta d'f *si $f \circ g = \mathrm{id}_Y$. També es pot definir el concepte d'inversa per l'esquerra de manera anàloga.*

TEOREMA 3.7. *L'Axioma d'Elecció 3.2 és equivalent a que tota funció sobrejectiva té inversa per la dreta.*

DEMOSTRACIÓ. En primer lloc demostrarem que l'Axioma d'Elecció implica que tota funció sobrejectiva té inversa per la dreta.

Considerem $f : X \longrightarrow Y$ una funció sobrejectiva. Aleshores definim el següent conjunt

$$\mathcal{X} = \{f^{-1}[\{y\}] \in \mathcal{P}(X) \ : \ y \in Y\} \tag{14}$$

El nostre objectiu és aplicar l'Axioma d'Elecció sobre este conjunt. Per a simplificar la demostració i gràcies al que hem demostrat al Teorema 3.4, només necessitem corroborar que $\mathcal{X}$ no continga al conjunt buit.

Fem la demostració per reducció a l'absurd. Suposem que el conjunt buit està contingut en $\mathcal{X}$. Això implicaria que, per definició, existeix un y en Y tal que $f^{-1}[\{y\}]$ és buit. No obstant, això no és possible ja que f és sobrejectiva i, per tant, tot element de Y té preimatge.

Per tant, podem aplicar l'Axioma d'Elecció i obtenim l'existència d'una funció d'elecció:

$$F : \mathcal{X} \longrightarrow \bigcup \mathcal{X}.$$

A partir d'aquesta funció podem definir la següent

$$\begin{array}{rccc} g: & Y & \longrightarrow & X \\ & y & \longmapsto & F\left(f^{-1}[\{y\}]\right) \end{array} \tag{15}$$

Aquesta funció està ben definida perquè la unió de $\mathcal{X}$ és un subconjunt de X i per definició de $\mathcal{X}$, $f^{-1}[\{y\}]$ està contingut.

Només falta demostrar que $f \circ g = \mathrm{id}_Y$. Donat y un element qualsevol de Y, tenim que $g(y)$ està contingut en $f^{-1}[\{y\}]$ en ser F una funció d'elecció. Això implica que $f(g(y))$ pertany a $\{y\}$. Així doncs, $(f \circ g)(y)$ és igual a y. Per tant, hem demostrat que g és una inversa per la dreta de f.

A continuació demostrem que si tota funció sobrejectiva té inversa per la dreta, aleshores es compleix l'Axioma d'Elecció.

Suposem que tenim un conjunt de conjunts $\mathcal{X}$ que són disjunts dos a dos i que no contenen al conjunt buit. Definim la funció següent:

$$G : \bigcup \mathcal{X} \longrightarrow \mathcal{X} \tag{16}$$

que a cada element x de $\bigcup \mathcal{X}$ li fa correspondre el conjunt X de $\mathcal{X}$ que el conté. Aquesta funció està ben definida ja que $\mathcal{X}$ és disjunt i per tant, per a cada x en $\bigcup \mathcal{X}$ existeix un únic X de $\mathcal{X}$ tal que x pertany a X.

El nostre objectiu és utilitzar la hipòtesi inicial, que totes les funcions sobrejectives tenen una inversa per la dreta, amb aquesta funció per a definir una funció d'elecció. Per fer-ho, primer necessitem corroborar que G és sobrejectiva. Això és evident, ja que, per a cada conjunt X de $\mathcal{X}$, com per hipòtesi no és buit, conté a algún element x que, per definició, és element de $\bigcup \mathcal{X}$ i per tant, $G(x) = X$. Així doncs, podem aplicar la hipòtesi de que tota funció sobrejectiva té inversa per la dreta, i per tant, anomenem F a aquesta inversa per la dreta. Notem que $F : \mathcal{X} \longrightarrow \bigcup \mathcal{X}$.

Només queda demostrar que F és una funció d'elecció, és a dir, que $F(X)$ és un element de X per a tot X de $\mathcal{X}$.

Per a demostrar-ho, considerem X un conjunt de $\mathcal{X}$. D'una banda, per definició de F, tenim que

$$(G \circ F)(X) = X. \tag{17}$$

D'altra banda, com $F(X)$ és un element de $\bigcup \mathcal{X}$, existeix un conjunt Y de $\mathcal{X}$ tal que $F(X)$ està contingut en Y. Aleshores, per construcció de G, sabem que

$$G(F(X)) = Y. \tag{18}$$

A partir de les equacions 17 i 18 obtenim que X és igual a Y i com a conseqüència, per definició de Y obtenim que $F(X)$ és un element de X, com volíem demostrar.

Amb això queda demostrada l'equivalència entre l'Axioma d'Elecció i que tota funció sobrejectiva té inversa per la dreta. □

NOTA 3.8. *Per a demostrar aquesta equivalència, hem fet servir el Teorema 3.4 que vam demostrar a la secció anterior. No obstant, es pot*

resoldre la demostració de forma independent utilitzant únicament la demostració de que el conjunt construït 14 és disjunt. Aquesta disjunció es deu al fet que, com es tracten d'antiimatges d'elements, en cas d'haver algun element que pertanyera a les antiimatges de dos elements diferents, significaria que aquest element tindria dos imatges distintes, és a dir, que estaria mal definit. Per tant, necessàriament hauria de ser disjunt.

3. Lema de Zorn-Kuratowski

En aquesta secció, explorarem una relació fonamental en la teoria de conjunts que connecta l'Axioma d'Elecció amb el Lema de Zorn-Kuratowski.

El Lema de Zorn-Kuratowski, conegut senzillament com el Lema de Zorn, és una eina fonamental que permet establir l'existència d'elements maximals en conjunts parcialment ordenats. Aquest resultat, aparentment modest, té ramificacions profundes en una àmplia gamma de disciplines matemàtiques.

La importància del Lema de Zorn-Kuratowski es fa evident quan considerem les seues aplicacions. En la teoria de grups i anells ajuda a demostrar la existència de subgrups o ideals maximals. A l'anàlisi funcional és crucial per a la construcció de bases en espais vectorials de dimensió infinita. En general, el Lema de Zorn-Kuratowski ens permet resoldre problemes que impliquen la selecció d'elements maximals en conjunts ordenats, i aquesta capacitat té un impacte profund en la resolució de problemes matemàtics diversos.

En aquesta secció, començarem en la definició de diversos conceptes que ens seran necessaris per a introduïr el lema.

La demostració d'una de les implicacions es basa principalment en la demostració realitzada a [**Lew91**]. D'altra banda, l'altra implicació es pot trobar en [**Buk19**].

Definició 3.9 (Ordre parcial). *Un ordre parcial sobre un conjunt X, és una relació binària $\leq$ definida en este conjunt que compleix les següents condicions:*

(1) *(Reflexivitat):* $\forall x \in X,\ (x \leq x)$.
(2) *(Antisimetria):* $\forall x, y \in X,\ ((x \leq y \wedge y \leq x) \longrightarrow x = y)$.
(3) *(Transitivitat):* $\forall x, y, z \in X,\ ((x \leq y \wedge y \leq z) \longrightarrow x \leq z)$.

Un conjunt parcialment ordenat és la tupla $(X, \leq)$, on X és un conjunt i $\leq$ un ordre parcial sobre X.

DEFINICIÓ 3.10 (Ordre estricte). *Un ordre estricte sobre un conjunt X és una relació binària $<$ definida en este conjunt que compleix les següents condicions:*

(1) *(Irreflexivitat):* $\forall x \in X,\ (x \not< x)$.
(2) *(Transitivitat):* $\forall x, y, z \in \mathcal{X},\ ((x < y \wedge y < z) \longrightarrow x < z)$.

En tot conjunt parcialment ordenat $(X, \leq)$, existeix una relació d'ordre estricte implícita definida com segueix:

$$\forall x, y \in X, (x < y \longleftrightarrow x \leq y \wedge x \neq y). \tag{19}$$

A partir d'aquest punt, per fer referència a una relació d'ordre estricte, utilitzarem les expressions "x és anterior a y"o "y és posterior a x"quan $x < y$.

DEFINICIÓ 3.11 (Cadena). *Una cadena, o conjunt totalment ordenat, és un conjunt C amb una relació d'ordre parcial $\leq$ en la qual tota parella d'elements està relacionada, és a dir, compleix la següent propietat:*

$$\forall x, y \in C,\ (x \leq y\ \vee\ x \geq y)\,. \tag{20}$$

A partir d'aquest punt, direm que $(X, \leq)$ és un conjunt parcialment ordenat i Y un subconjunt de X.

DEFINICIÓ 3.12 (Fita superior). *Un element s de X és una fita superior de Y si satisfà la següent condició:*

$$\forall y \in Y,\ (y \leq s). \tag{21}$$

Denotem per $\mathrm{Fsup}_X(Y)$ *al conjunt de totes les fites superiors de Y en X. Direm que un fita superior és estricta si es satisfà la condició amb ordre estricte.*

DEFINICIÓ 3.13 (Suprem). *Direm que un element s és un suprem d'un conjunt Y en X si compleix les següents condicions:*

(1) *s és fita superior de Y en X, i.e.,* $s \in \mathrm{Fsup}_X(Y)$.
(2) *Per a cada* $x \in \mathrm{Fsup}_X(Y), s \leq x$.

DEFINICIÓ 3.14 (Element maximal). *Direm que m és un element maximal de X si compleix la següent propietat:*

$$\forall x \in X,\ \ (m \leq x \longrightarrow m = x).$$

DEFINICIÓ 3.15 (Mínim). *Siga z un element de X, direm que z és un mínim de X si satisfà la següent propietat:*

$$\forall x \in X,\ \ (z \leq x).$$

Ja estem en condicions de poder definir el Lema de Zorn-Kuratowski.

LEMA 3.16 (Lema de Zorn-Kuratowski). *Siga $(X, \leq)$ un conjunt parcialment ordenat no buit per al què tota cadena no buida en X té suprem, aleshores existeix un element maximal de X.*

Per a poder demostrar l'equivalència del Lema de Zorn-Kuratowski i l'Axioma d'Elecció introduïm els següents conceptes.

DEFINICIÓ 3.17 (Segment inicial d'un conjunt fins a un element). *Siga $(X, \leq)$ un conjunt ordenat, C una cadena en X i x un element de C, definim $C_{\downarrow x}$ com al conjunt format per tots els elements de C anteriors a x. La definició formal seria la següent:*

$$C_{\downarrow x} = \{y \in C \ : \ y < x\}. \tag{22}$$

DEFINICIÓ 3.18 (Fites superiors estrictes d'un conjunt). *Siga $(X, \leq$ $)$ un conjunt ordenat, C una cadena en X, definim* $\mathrm{Upp}(C)$ *com el conjunt que conté totes les fites superiors estrictes de C. La definició formal és la següent:*

$$\mathrm{Upp}(C) = \{x \in X \ : \ \forall y \in C, (y < x)\}. \tag{23}$$

Amb açò estem ara preparats per a procedir amb la demostració de l'equivalència.

TEOREMA 3.19. *L'Axioma d'Elecció 3.2 implica el Lema de Zorn-Kuratowski 3.16.*

DEMOSTRACIÓ. Comencem demostrant que l'Axioma d'Elecció implica el Lema de Zorn-Kuratowski.

Considerem $(X, \leq)$ un conjunt parcialment ordenat no buit en el què tota cadena no buida té un suprem, i volem demostrar que hi ha un element maximal en $(X, \leq)$.

Raonarem per reducció a l'abasurd. Suposem que no hi ha cap element maximal en X. A partir d'aquesta suposició, demostrarem el següent lema intermedi.

LEMA 3.20 (Lema A). *Tota cadena C de X, té fites superiors estrictes, és a dir, el conjunt* $\mathrm{Upp}(C)$ *no és buit.*

DEMOSTRACIÓ (LEMA A). En primer lloc, considerem el cas en què C és buit. En aquest cas, segons la definició de $\mathrm{Upp}(\emptyset)$, aquest conjunt és igual a tot X. Com hem suposat que X no és buit, ja hem demostrat el lema en aquest cas.

Ara bé, si C no és buit, com hem assumit que cada cadena en X té un suprem, podem afirmar que existeix un suprem p de C.

Sobre p considerem els següents casos. En primer lloc, si p pertany a C, aleshores, si existeix algun element x posterior a p (és a dir, $p < x$),

tenim que x pertany a $\mathrm{Upp}(C)$. En cas contrari, si no hi ha cap element posterior a p, aleshores p és un element maximal, la qual cosa entra en contradicció amb la nostra suposició inicial de que no hi ha elements maximals en $(X, \leq)$.

D'altra banda, si p no pertany a C, aleshores p està en $\mathrm{Upp}(C)$. Per tant, en ambdós casos, $\mathrm{Upp}(C)$ no pot ser buit, i així s'ha demostrat el Lema (A). □

Continuant amb la demotració, definim el següent conjunt

$$\mathcal{C} = \{\mathrm{Upp}(C) \in \mathcal{P}(\mathcal{X}) \ : \ C \text{ cadena}\}. \tag{24}$$

Ara bé, pel Lema A 3.20, sabem que $\mathcal{C}$ no conté al buit i, per tant, podem aplicar l'Axioma d'Elecció, ja que, aquesta aplicació no depén de si és disjunt, com es demostra al Teorema 3.4. Així, obtenim una funció d'elecció:

$$F : \mathcal{C} \longrightarrow \bigcup \mathcal{C} \tag{25}$$

que compleix $F(\mathrm{Upp}(C)) \in \mathrm{Upp}(C)$ per a tota cadena C. En altres paraules, per a cada cadena C, $F(\mathrm{Upp}(C))$ és una fita superior de C no continguda en C.

Ara definim el següent conjunt auxiliar.

Definició 3.21 (Conjunt Conforme). *Siga A un conjunt no buit en X, direm que és* conforme *si compleix les següents propietats:*

(1) *Tot subconjunt no buit d'A té un element mínim. Aquesta propietat determina que A és una cadena, ja que per a tots dos elements x i y d'A, existeix el conjunt $\{x, y\}$, el qual, per aquesta propietat, té un element mínim. Això implica que o bé $x \leq y$ o bé $x \geq y$, per tant A és una cadena. Així, la següent propietat està ben definida.*

(2) *Per a tot element x d'A, es compleix la següent igualtat*

$$x = F(\mathrm{Upp}(A_{\downarrow x})). \tag{26}$$

Per a continuar, demostrem el següent lema auxiliar.

Lema 3.22 (Lema B). *Donats dos conjunts A i B conformes tal que $A \setminus B$ no és buit, aleshores existeix un x de $A \setminus B$ tal que $B = A_{\downarrow x}$.*

Demostració (Lema B). Com que $A \setminus B$ no és buit, és un subconjunt d'A i A és conforme, podem afirmar que existeix un element mínim d'aquest conjunt $A \setminus B$, al qual anomenarem x. El nostre objectiu és demostrar que $A_{\downarrow x} = B$.

Per demostrar-ho, comencem amb la inclusió $A_{\downarrow x} \subseteq B$. Suposem que w és un element de $A_{\downarrow x}$ i suposem que w no pertany a B. Això implica que w està en $A \setminus B$, i com que x és mínim de $A \setminus B$, obtenim

que $w < x \leq w$ el que és una contradicció, que s'obté de suposar que w no es un element de B.

Ara, demostrem la inclusió contrària, $A_{\downarrow x} \supseteq B$. Siga w un element de B, suposem, per contradicció, que w no és un element de $A_{\downarrow x}$. En aquest cas, tenim que $B \setminus A_{\downarrow x}$ és un subconjunt de B no buit. Aleshores, com B és conforme, existeix un element mínim de $B \setminus A_{\downarrow x}$ que denotarem com y.

Per continuar, demostrem el següent lema intermedi.

LEMA 3.23 (Lema B.1). *Donat un element v d'A i un element u de $B_{\downarrow y}$ de manera que $v < u$, aleshores, v és un element de $B_{\downarrow y}$.*

DEMOSTRACIÓ LEMA B.1. En primer lloc, s'observa directament que $v < u < y$. Així, l'única qüestió pendent és demostrar que v és un element de B. Per a aconseguir-ho, suposem que v no pertany a B. Aleshores tenim que v és un element de $A \setminus B$. Com que x és el mínim d'aquest conjunt, tenim que $x \leq v < u$. Això implica que u no pertany a $A_{\downarrow x}$ i, per tant, u és un element de $B \setminus A_{\downarrow x}$. Així doncs, tenim que $y \leq u$. No obstant, com u és un element de $B_{\downarrow y}$, tenim que $u < y \leq u$, la qual cosa és una contradicció deguda a que hem suposat que v no és un element de B. Així, queda demostrat el lema intermedi. □

Continuem amb la demostració del Lema B, com que $A \setminus B$ no és buit, tampoc ho és $A \setminus B_{\downarrow y}$. Aleshores, com que A és conforme, existeix un element mínim de $A \setminus B_{\downarrow y}$, que denotarem z.

Per concloure la demostració del Lema B, només cal demostrar un últim lema intermedi.

LEMA 3.24 (Lema B.2). *Es té la següent igualtat de conjunts*

$$A_{\downarrow z} = B_{\downarrow y}.$$

DEMOSTRACIÓ LEMA B.2. Comencem la demostració mostrant que $A_{\downarrow z} \subseteq B_{\downarrow y}$. Siga w un element de $A_{\downarrow z}$ i, per tant, element d'A. Suposem, que w no pertany a $B_{\downarrow y}$. Això implica que w és un element de $A \setminus B_{\downarrow y}$. Com z és el mínim d'este conjunt, obtenim que $z \leq w < z$. Hem arribat a una contradicció, degut a que hem suposat que w no és un element de $B_{\downarrow y}$. Per tant, queda demostrada la inclusió $A_{\downarrow z} \subseteq B_{\downarrow y}$.

Només queda demostrar la inclusió contrària. Siga w un element de $B_{\downarrow y}$. Suposem que w no és un element d'A. Això implica que w tampoc pertany a $A_{\downarrow x}$. Com a conseqüència, obtenim que w és un element de $B \setminus A_{\downarrow x}$. Per tant $y \leq w$. Açò entra en contradicció amb la hipòtesi inicial, que diu que $w < y$. Aquesta contradicció es deu al fet que hem suposat que w no està en A. Per tant, necessàriament s'ha de complir que $w \in A$.

Per acabar amb la demostració, només necessitem demostrar que $w < z$. Suposem que no es compleix. Com que A és una cadena (en ser conforme), tenim que $z \leq w$. Per tant, hi ha dues possibilitats: $z = w$ o $z < w$.

En el cas de suposar que $z = w$, z no és un element de $B_{\downarrow y}$ mentre w si ho és, per tant hem arribat a una contradicció i no podem considerar aquesta possibilitat. Si suposem $z < w$, pel Lema B.1 3.23 obtenim que z és un element de $B_{\downarrow y}$. Això també és impossible per definició de z. Pel que, necessàriament s'ha de satisfer la condició de que $w < z$. Per tant, obtenim que w és un element de $A_{\downarrow z}$. Amb açò quedaria demostrat el Lema B.2. □

Per concloure la demostració del Lema B, utilitzarem el Lema B.2 3.24 i la segona condició que defineix conjunts conformes, obtenint la següent igualtat:

$$y = F(\mathrm{Upp}(B_{\downarrow y})) = F(\mathrm{Upp}(A_{\downarrow z})) = z. \tag{27}$$

Ara, com x és un element d'$A \setminus B$, també és un element d'$A \setminus B_{\downarrow y}$ pel que, $y = z \leq x$. A més, com x no pertany a B per definició, tenim que $y \neq x$, la qual cosa implica que $y < x$. Això ens porta a la conclusió de que y és un element d'$A_{\downarrow x}$. No obstant, aquesta conclusió entra en contradicció amb la definició de y. Aquesta contradicció ve donada pel fet que hem suposat que w no és un element de B. Amb açò, queda demostrat el Lema B. □

Continuem en la demostració amb el següent lema intermedi

Lema 3.25 (Lema C). *Siga $\mathcal{A}$ un conjunt de conjunts conformes no buit, aleshores $\bigcup \mathcal{A}$ és conforme.*

Demostració (Lema C). En primer lloc, tenim que $\bigcup \mathcal{A}$ no és buida. Com $\mathcal{A}$ no és buida, existeix un conjunt A d'$\mathcal{A}$ que és conforme i, per tant, no buit.

En segon lloc, considerem un subconjunt Y de $\bigcup \mathcal{A}$ no buit. Demostrem que aquest conjunt té mínim. Com que Y no és buit, aleshores existeix un element a en Y i un conjunt A d'$\mathcal{A}$ tal que a és un element d'A. Ara bé, com que A és conforme i $Y \cap A$ és un subconjunt no buit d'A, obtenim que aquest conjunt té un mínim que anomenem z. El nostre objectiu és demostrar que aquest z és el mínim de Y. Per aconseguir-ho, prenim un element b en Y. Si b és un element d'A, aleshores $z \leq b$. Si b no és un element d'A, aleshores existeix un conjunt B d'$\mathcal{A}$ tal que b és un element de B. Per tant, obtenim que $B \setminus A$ no és buit, el que implica, pel Lema B 3.22), que existeix un x de $B \setminus A$ tal que $A = B_{\downarrow x}$ i per tant, $z < x \leq b$. Així, hem demostrat que z és el mínim de Y, com volíem demostrar.

Per acabar, només cal demostrar la segona propietat de conjunts conformes. Siga a un element de $\bigcup \mathcal{A}$. Aleshores existeix un conjunt A d'$\mathcal{A}$ tal que a és un element d'A. Aleshores $A_{\downarrow a} = (\bigcup \mathcal{A})_{\downarrow a}$. Això ve donat perquè, si considerem un element w de $(\bigcup \mathcal{A})_{\downarrow a}$ i suposem que w no pertany a A, aleshores existeix un conjunt B de $\bigcup \mathcal{A}$, tal que w és un element de B. Això implica que $B \setminus A$ no és buit. Per tant, pel Lema B 3.22, existeix un element x, mínim de $B \setminus A$, tal que $A = B_{\downarrow x}$. Però aleshores obtenim el següent:

$$a < x \leq w < a. \tag{28}$$

Aquesta contradicció ve de suposar que w no és un element d'A. Per tant, tenim que $A_{\downarrow a} = (\bigcup \mathcal{A})_{\downarrow a}$ i, així, per la segona propietat de conjunts conformes, obtenim:

$$a = F\left(\mathrm{Upp}\left(A_{\downarrow a}\right)\right) = F\left(\mathrm{Upp}\left(\left(\bigcup \mathcal{A}\right)_{\downarrow a}\right)\right). \tag{29}$$

Amb això, queda demostrada la segona propietat de conjunts conformes i, per tant, el Lema C està demostrat. □

Per finalitzar la demostració de la implicació, definim el conjunt:

$$\mathcal{K} = \{A \in \mathcal{P}(X) : A \text{ conforme}\}. \tag{30}$$

D'una banda, observem que $\mathcal{K}$ no és buit, ja que, podem definir w de la següent forma:

$$w = F(\mathrm{Upp}(\emptyset)) = F(\mathrm{Upp}(\{w\}_{\downarrow w})). \tag{31}$$

Això ens assegura que w és un element de X, i per tant, el conjunt $\{w\}$ és conforme.

Aleshores, utilitzant el Lema C 3.25 deduim que $\bigcup \mathcal{K}$ és un conjunt conforme. Ara bé, siga:

$$k = F\left(\mathrm{Upp}\left(\bigcup \mathcal{K}\right)\right). \tag{32}$$

Tenim que k és un element posterior a $\bigcup \mathcal{K}$, el que implica que k no és element de $\bigcup \mathcal{K}$. Però aleshores, considerem el conjunt $\{k\} \cup (\bigcup \mathcal{K})$. Aquest és un conjunt conforme que no està contingut a $(\bigcup \mathcal{K})$. No obstant, com és conforme, ha d'estar en $\mathcal{K}$ i per tant ha de ser subconjunt de $(\bigcup \mathcal{K})$. Això ens porta a una contradicció, que ve donada per suposar que no existeix un element maximal de X.

Amb aquesta conclusió, hem demostrat que l'Axioma d'Elecció implica el Lema de Zorn-Kuratowski. □

Ara procedim a demostrar que el Lema de Zorn-Kuratowski implica l'Axioma d'Elecció.

TEOREMA 3.26. *El Lema de Zorn-Kuratowski 3.16 implica l'Axioma d'Elecció 3.2.*

DEMOSTRACIÓ. Siga $\mathcal{X}$ un conjunt de conjunts disjunts dos a dos que no conté al buit, volem demostrar que existeix una funció d'elecció definida en este conjunt.

Si $\mathcal{X}$ és un conjunt buit, l'aplicació buida, o el que és el mateix, el conjunt buit, és una funció d'elecció, per tant ja tenim demostrat l'Axioma d'Elecció en aquest cas.

Ara, si suposem que $\mathcal{X}$ no és buit, aleshores definim el següent conjunt:

$$P = \left\{ (\mathcal{Y}, G) \in \mathcal{P}(\mathcal{X}) \times \mathrm{Fun}(\mathcal{Y}, \bigcup \mathcal{Y}) : \forall Y \in \mathcal{Y}, (G(Y) \in Y) \right\}. \quad (33)$$

Sobre aquest conjunt definim l'ordre parcial $\leq$ següent:

$$(\mathcal{Y}, G) \leq (\mathcal{Z}, H) \longleftrightarrow \mathcal{Y} \subseteq \mathcal{Z} \wedge G = H_{\restriction \mathcal{Y}}. \quad (34)$$

Amb aquesta relació, $(P, \leq)$ és un conjunt parcialment ordenat. A continuació comprovem sobre aquest conjunt ordenat podem aplicar el Lema de Zorn-Kuratowski.

En primer lloc, com que hem suposat que $\mathcal{X}$ no és buit, sabem que existeix un X de $\mathcal{X}$ que és no buit per hipòtesi i, per tant, existeix un x element de X. Pel que podem definir la següent funció:

$$\begin{array}{rccc} J: & \{X\} & \longrightarrow & \bigcup\{X\} \\ & X & \longmapsto & x \end{array} \quad (35)$$

De manera que (X, J) és un element de P. Aleshores, P no és buit.

D'altra banda, siga $\mathcal{C}$ una cadena de $(P, \leq)$, definim

$$\mathcal{Z} = \bigcup_{(\mathcal{Y}, G) \in \mathcal{C}} \mathcal{Y} = \{Y \in \mathcal{X} : \exists (\mathcal{Y}, G) \in \mathcal{C}, (Y \in \mathcal{Y})\}. \quad (36)$$

Aleshores, donat un element $X \in Z$, definim

$$\mathrm{Upp}_{\mathcal{C}}(X) := \{G : \exists (\mathcal{Y}, G) \in \mathcal{C}, X \in \mathcal{Y}\}. \quad (37)$$

Tal que, per a tot $G \in \mathrm{Upp}_{\mathcal{C}}(X)$, $G(X)$ és única. Amb açò podem definir l'aplicació

$$\begin{array}{rccc} \mathcal{G} & : & \mathcal{Z} & \longrightarrow & \bigcup \mathcal{Z} \\ & & X & \longmapsto & G(X) \end{array} \quad (38)$$

Per tant, tenim que $(\mathcal{Z}, \mathcal{G})$ és un suprem de $\mathcal{C}$.

Aleshores podem aplicar el Lema de Zorn-Kuratowski a $(P, \leq)$ i obtenim que existeix un element maximal de P, anomenem-lo $(\mathcal{M}, F)$. Només queda demostrar que $\mathcal{M}$ és igual a $\mathcal{X}$ per a finalitzar la demostració. Per a fer-ho, suposem que són distints, per tant existeix un

conjunt X de $\mathcal{X} \setminus \mathcal{M}$, no buit, pel que podem considerar x un element de X i per tant definim la següent aplicació:

$$\begin{array}{rccl} G: & \mathcal{M} \cup \{X\} & \longrightarrow & \bigcup(\mathcal{M} \cup \{X\}) \\ & Y & \longmapsto & \begin{cases} F(Y) & \text{si } Y \in \mathcal{M}; \\ x & \text{si } Y = X. \end{cases} \end{array} \tag{39}$$

Així definida, hem obtés una funció d'elecció del conjunt $\mathcal{M} \cup \{X\}$, de manera que el conjunt $(\mathcal{M} \cup \{X\}, G)$ és un element de P i $(\mathcal{M}, F) < (\mathcal{M} \cup \{X\}, G)$, la qual cosa contradiu el fet de que $(\mathcal{M}, F)$ és un element maximal. Aquesta contradicció ve de suposar que $\mathcal{M}$ és distint de $\mathcal{X}$. Per tant $\mathcal{M} = \mathcal{X}$ i F és una funció d'elecció per a $\mathcal{X}$. □

TEOREMA 3.27. *L'Axioma d'Elecció 3.2 és equivalent al Lema de Zorn-Kuratowski 3.16.*

DEMOSTRACIÓ. Se segueix dels Teoremes 3.19 i 3.26. □

CAPíTOL 4

Formalització de la demostració en Lean

En aquest capítol, introduirem els conceptes fonamentals de Lean, un llenguatge de programació funcional i verificador formal, i explorarem la Teoria de Tipus. També descriurem com hem establert les equivalències de l'Axioma d'Elecció, que es poden trobar en línia [**PL23**].

1. Lean: Un verificador formal

Lean és una poderosa ferramenta en el camp de les matemàtiques i la lògica formal que busca entrellaçar les ferramentes formals interactives amb les automàtiques, a més de proporcionar una base sòlida i rigorosa per a la verificació de demostracions i sistemes complexos [**DM15**].

A un nivell fonamental, Lean es basa en la teoria de tipus dependents, que permet especificar amb precisió l'estructura i el comportament dels objectes matemàtics. Això facilita la creació de definicions formals, lemes i teoremes, la qual cosa permet verificar la correcció d'algorismes i demostracions de manera sistemàtica [**ADMK17**].

A més, Lean és una eina de codi obert, la qual cosa significa que és accessible per a la comunitat científica i de desenvolupament de programari. Això ha portat a la creació d'una àmplia biblioteca de teoremes verificats [**LC**] i una comunitat activa d'usuaris que col·laboren en el seu desenvolupament i aplicació en diversos camps, des de les matemàtiques pures fins a la intel·ligència artificial [**Ope22**].

A banda, una de les característiques destacables de Lean, que el distingeix com una eina de formalització poderosa, és la seua capacitat de compilació en temps real i la seua capacitat de mostrar de manera simultània l'estat actual del treball i els objectius que cal aconseguir per a realitzar una demostració. Aquest atribut és de suma importància i ofereix una avantatge significativu en el context de la formalització i la verificació de teoremes [**DM15**].

En resum, Lean és una eina potent de formalització que combina matemàtiques i programació per verificar la correcció de sistemes i programes. El seu enfocament en la teoria de tipus dependents, i el seu caràcter de codi obert el converteixen en una opció atractiva per a

aquelles persones que busquen garantir la fiabilitat de sistemes crítics i demostrar resultats matemàtics de manera rigorosa.

1.1. teoria de tipus. La teoria de tipus és una part fonamental de Lean i moltes altres ferramentes de formalització. En aquesta secció, introduirem els conceptes bàsics de la teoria de tipus, incloent la idea de tipus dependents i com s'utilitzen per a garantir la consistència i la correcció en les formalitzacions matemàtiques.

El matemàtic Bertrand Russell, del qual hem parlat abans, va ser un dels pioners en aquesta teoria, que va desenvolupar en la seua obra "Principia Mathematica", juntament amb Alfred N. Whitehead [**WR27**].

La motivació de Russell per a crear aquesta teoria va ser resoldre la paradoxa que porta el seu nom, de la qual hem introduït al Capítol 1, que mostrava una inconsistència en la teoria de conjunts. Per a aconseguir-ho, Russell va introduïr una jerarquia de nivells lògics, asignant cada entitat matemàtica, com els nombres, les funcions o els conjunts, a un tipus. On els objectes d'un tipus donat només poden ser creats per un tipus anterior, evitant així els bucles i les paradoxes [**WR27**].

La importància de la teoria de tipus en la formalització informàtica prové de diversos aspectes:

(1) **Resolució de paradoxes**: La teoria de tipus va ajudar a resoldre les paradoxes que van posar en perill els fonaments de les matemàtiques i la lògica. La noció de tipus permet evitar les autoreferències i les contradiccions que van sorgir en la teoria del conjunts.
(2) **Precisió en la computació**: En informàtica, la teoria de tipus és fonamental per a garantir la correcció i la seguretat dels programes. Assegura que les operacions s'apliquen a valors compatibles i ajuda a evitar errors comuns, com ara els errors de tipus [**Bar08**].
(3) **Formalització matemàtica**: La teoria de tipus també és crucial en la formalització matemàtica. Permet la representació precisa de les estructures matemàtiques i la formalització de les demostracions matemàtiques en sistemes assistits per ordinador com Lean i Coq [**BDS13**].

2. Formalització en Lean

Ara que hem introduït els fonaments de Lean, estem preparats per a començar a definir els conceptes matemàtics utilitzant aquesta ferramenta. En les pròximes seccions del TFG, explorarem com Lean ens

permet formalitzar i raonar sobre els conceptes que són rellevants per a la nostra investigació.

2.1. Metodologia. Una de les característiques destacades de Lean és el seu enfocament en la formalització incremental. Els matemàtics i verificadors poden construir gradualment definicions, lemes i teoremes a mesura que avancen en la seua feina. Lean compila i verifica cada pas en temps real, el que significa que els errors es detecten i corregeixen immediatament, en lloc d'esperar fins al final del procés.

Cal tindre en compte que un dels principals avantatges que ens proporciona Lean és l'ús del llenguatge matemàtic al codi, ja que, operadors com $\neg$, $\wedge$, $\vee$, $\exists$, $\forall$, $\in$, $\longrightarrow$, $\leftrightarrow$, $\subseteq$ i $\leq$ són operadors a Lean, que fan la mateixa funció que a la matemàtica en paper. Aquest fet ens proporcionen una comprensió molt més clara i visual del que s'està treballant.

2.1.1. *Exemple: Formalització del Modus Tollens.* Suposem que volem formalitzar la proposició lògica de Modus Tollens:

```
1 lemma MT (P Q: Prop)(h : P⟶Q) : (¬Q⟶¬P):=
2 begin
3   intros notQ,
4   by_contradiction hP,
5   have hQ:Q,from h hP,
6   exact notQ hQ,
7 end
```

En aquesta demostració, es declaren dues proposicions `P` i Q (que són afirmacions o enunciats lògics) com a variables, i s'estableix una hipòtesi addicional `h` del tipus $P \longrightarrow Q$.

Definim el marc de la demostració amb els termes `begin` i `end`, que denoten l'inici i final de la demostració. En aquest cas, el nostre objectiu és demostrar que si assumim $\neg Q$, llavors podem demostrar $\neg P$.

Aleshores, s'utilitza la comanda `intros notQ` per a introduir la hipòtesi $\neg Q$ i emmagatzemar-la a una variable que hem anomenat `notQ`. El que estem fent és suposar $\neg Q$, i per tant, el notre objectiu deixa de ser la implicació original i canvia a $\vdash$ `¬P`.

La comanda `by_contradiction hP` inicia una suposició contradictòria. Això significa que assumim el contrari del que volem demostrar, que com en aquest cas volíem demostrar $\neg P$, obtenim `hp:P` i el nostre objectiu canvia a arribar a una contradicció, que al programa s'expressa com $\vdash$ `false`.

A continuació, a la línia 5, fem ús de `have hQ:Q,from h hP`. En primer lloc, la comanda `have hQ:Q,` proporciona un nou objectiu dins

de la demostració. La comanda `from h hP`, el que fa és, obtindre, de la hipòtesi `h:(P⟶Q)` i `hP:P`, el resultat `Q`. A més, aquest resultat es guarda a la variable que hem nomenat `hQ`.

Finalment a la línia `exact notQ hQ`, la comanda `exact` estableix que s'ha arribat al resultat, i el resultat que obtenim és una contradicció que prové de `notQ hQ`, ja que, el funcionament de Lean de l'operador ¬ està definit com

$$\neg Q \equiv Q \longrightarrow \texttt{false}. \tag{40}$$

Per tant, si apliquem Q a la proposició $\neg Q$ obtenim `false`, es a dir, el resultat que volíem obtindre.

2.2. Teoria axiomàtica de Conjunts en Lean. En Lean, per a utilitzar la teoria axiomàtica de conjunts, farem servir una biblioteca anomenada `set_theory`, la qual defineix els conceptes necessaris per a treballar en aquest marc teòric. Lean, per defecte, està orientat cap a una teoria de tipus, i definir conceptes com ara aplicacions de manera directa pot ser diferent de com ho faríem en una teoria de conjunts.

En la biblioteca `set_theory`, el tipus `Set` és el concepte central que defineix i representa els conjunts. Aquest tipus s'utilitza per a crear i definir conjunts, i està fonamentat en els axiomes i les regles de la teoria de conjunts. A través del tipus `Set`, mitjançant els axiomes de (ZF), es poden formalitzar propietats, relacions i operacions que involucren conjunts de manera precisa.

Altre concepte a destacar en aquesta llibreria serien els universos. En la teoria de tipus en la que es basa Lean, un univers és un tipus que conté a altres tipus i que compleix certes propietats que permeten formar nous tipus a partir dels existents. Els universos s'utilitzen per a evitar l'aparició de paradoxes en la teoria de tipus, entre altres coses permeten definir tipus que contenen a tots els demés tipus el que permet definir tipus recursius [**ML80**]. La qüestió que obtenim amb la teoria de conjunts és que, per a poder utilitzar aplicacions o realitzar demostracions amb els elements de tipus `Set` necessitem que, en tot moment, els elements es troben en el mateix univers. En cas contrari, per com està construït aquest tipus, podem arribar a que hem realitzat una demostració correcta, però, en no estar definida al mateix univers, no es done per vàlida. Al llarg de tot aquest capítol anem a denotar

```
1 universe u
```

com a l'univers que comparteixen totes les definicions i demostracions per a així evitar problemes.

En resum, mitjançant `set_theory`, Lean ens permet utilitzar la teoria axiomàtica de conjunts i definir conceptes de manera consistent amb

aquesta perspectiva, la qual cosa facilita la realització de demostracions i formalitzacions matemàtiques en aquest marc teòric específic.

Durant aquest capítol, farem servir la notació `[...]` per a marcar l'absència d'una demostració, que en tot cas es trobarà disponible als arxius de GitHub [**PL23**] o a la biblioteca de `set_theory` que podem trobar a [**LC**]. Aquesta notació servirà per a comentar de forma més senzilla el codi que presentarem.

2.2.1. *Definició dels Axiomes de Zermelo-Fraenkel en Lean.* A continuació procedim a exposar les definicions dels axiomes (ZF) en aquesta biblioteca:

```
--Extensionalitat
theorem ext_iff {x y : Set.{u}} :
(∀ z : Set.{u}, z ∈ x ↔ z ∈ y) ↔ x = y := [...]

--Existència del buit
def empty : Set := mk ∅
theorem not_mem_empty (x : Class.{u}) : x ∉ (∅ : Class.{u
    }) := [...]

--Conjunt parell no ordenat
theorem mem_pair {x y z : Set.{u}} :
x ∈ ({y, z} : Set) ↔ x = y ∨ x = z := [...]

--Conjunt unió
def sUnion : Set ⟶ Set:= [...]
theorem mem_sUnion {x y : Set.{u}} :
y ∈ sUnion x ↔ ∃ z ∈ x, y ∈ z := [...]

--Conjunt potència
def powerset : Set ⟶ Set := [...]
theorem mem_powerset : { x y : Set}, y ∈ powerset x ↔ y
    ⊆ x := [...]

--Separació
protected def sep (p : Set ⟶ Prop) : Set ⟶ Set :=
    [...]
theorem mem_sep {p : Set.{u} ⟶ Prop} {x y : Set.{u}} :
y ∈ {y ∈ x | p y} ↔ y ∈ x ∧ p y := [...]
```

Com es pot observar, els axiomes no es defineixen com a `axiom`, sinó com a `theorem`, això ve donat perquè aquesta teoria de conjunts està contruïda a partir de la teoria de tipus i per tant, aquests axiomes es poden obtindre a partir dels axiomes de la teoria de tipus sense necessitat de definir-los de nou.

Ara proporcionarem les definicions de diversos conceptes bàsics de teoria de conjunts en Lean, necessaris per a l'ús de de teoria de conjunts en aquest:

```
--Subconjunt (Definició (2.2))
def subset (x y : Set.{u}) := ∀ {{z}}, z ∈ x ⟶ z ∈ y
instance has_subset : has_subset Set := ⟨Set.subset⟩

--Conjunts amb un únic element
theorem mem_singleton {x y : Set.{u}} : x ∈ @singleton
   Set.{u} Set.{u} _ y ↔ x = y := [...]

--Unió entre dos conjunts (Definició (2.3))
protected def union (x y : Set.{u}) : Set.{u} := ⋃₀ {x, y
   }
instance : has_union Set := ⟨Set.union⟩
theorem mem_union {x y z : Set.{u}} : z ∈ x ∪ y ↔ z ∈ x
   ∨ z ∈ y := [...]

--Intersecció entre dos conjunts (Definició (2.4))
protected def inter (x y : Set.{u}) : Set.{u} := {z ∈ x |
    z ∈ y}
instance : has_inter Set := ⟨Set.inter⟩
theorem mem_inter {x y z : Set.{u}} : z ∈ x ∩ y ↔ z ∈ x
   ∧ z ∈ y := [...]

--Diferència entre dos conjunts (Definició (2.5))
protected def diff (x y : Set.{u}) : Set.{u} := {z ∈ x |
   z ∉ y}
instance : has_sdiff Set := ⟨Set.diff⟩
theorem mem_diff {x y z : Set.{u}} : z ∈ x \ y ↔ z ∈ x ∧
    z ∉ y := [...]
```

A continuació, comentarem altres definicions rellevants en Lean utilitzant la teoria de conjunts.

2.2.2. *Parell d'elements ordenats.* Direm que una tupla d'elements, o parell d'elements ordenats ve definit de la següent forma:

$$(x, y) = \{\{x\}, \{x, y\}\} \tag{41}$$

Això ve determinat en lean de la següent manera:

```
/-- Kuratowski ordered pair -/
def pair (x y : Set.{u}) : Set.{u} := {{x}, {x, y}}
```

Que és la definició que va donar Kuratowski de parell ordenat i que trobem a la Definició 2.6.

2.2.3. *Producte cartesià.* Ara bé, en teoria de conjunts, sabem que un producte de conjunts es defineix seguint la Definició 2.7.

Però a Lean, per comoditat, es construeix un cas més general:

```
def pair_sep (p : Set.{u} ⟶ Set.{u} ⟶ Prop) (x y : Set
    .{u}) : Set.{u} :=
{z ∈ powerset (powerset (x ∪ y)) |
    ∃ a ∈ x, ∃ b ∈ y, z = pair a b ∧ p a b}
```

A partir d'ací es defineix el producte cartesià entre dos conjunts de la següent manera:

```
def prod : Set.{u} ⟶ Set.{u} ⟶ Set.{u} := pair_sep (λ
    a b, true)
```

En aquest cas, el que ha ocorregut és que s'ha definit el concepte de l'esquema axiomàtic de separació per a cada parell d'elements ordenats, de manera que podem definir el producte de dos conjunts com a tots els parells ordenats tals que, el primer element està en X i el segon element està en Y sense cap condició.

2.2.4. *Definició de Funció.* En la biblioteca `set_theory`, les funcions, d'igual forma que a la Definició 2.8, en Lean es defineix la condició de que un conjunt siga funció de la següent forma.

```
def is_func (x y f : Set.{u}) : Prop :=
f ⊆ prod x y ∧ ∀ z : Set.{u}, z ∈ x ⟶ ∃!w, pair z w ∈ f
```

A més, d'igual forma a la Definició 2.8, podem trobar que en Lean està definit el conjunt de funcions entre dos conjunts.

```
def funs (x y : Set.{u}) : Set.{u} :=
{f ∈ powerset (prod x y) | is_func x y f}
```

2.2.5. *Composició de funcions.* Definim la composició de funcions en la teoria de conjunts com:

Donades les funcions $g : X \longrightarrow Y$ i $f : Y \longrightarrow Z$,

$$f \circ g = \{(a, c) \in X \times Z : \exists b \in Y, (a, b) \in g \wedge (b, c) \in f\} \tag{42}$$

Aleshores, transcrivint a Lean, obtenim la següent definició de composició de funcions:

```
def comp (hf:f ∈ (Y.funs Z))(hg:g ∈ (X.funs Y)):Set.{u}:=
{w∈ X.prod Z| ∃x y z : Set.{u}, (x.pair y) ∈g
∧ (y.pair z)∈f ∧ w = x.pair z}
infix ∘∘ := comp
```

A més, podem demostrar, fàcilment, que tota composició de funcions és una funció ben definida:

```
lemma comp_is_func (hf:f ∈ (Y.funs Z))(hg:g ∈ (X.funs Y))
    : (hf ∘∘ hg)∈X.funs Z:= [...]
```

2.2.6. *Conjunt disjunt.* Definim conjunt disjunt a partir de la definició de (3.1):

```
def disjunts:= a ∩ b = (∅:Set.{u})
def Disj (A:Set.{u}): Prop:= ∀ a b∈A, disjunts a b  ∨ a =
    b
```

Aleshores ja tenim tot el que necessitem per a començar les demostracions de les equivalències del capítol 3 mitjançant Lean.

3. Demostracions amb Lean

A continuacuó vorem com hem completat la formalització de les equivalències de l'Axioma d'Elecció que hem demostrat teòricament al capítol anterior utilitzant Lean. En aquesta secció procedirem a comparar els conceptes necessaris en Lean per a poder arribar a les demostracions sense posar-nos en detall en el desenvolupament de les mateixes. No obstant, la formalització completa la podem trobar al repositori de Github [**PL23**].

A continuació, procedirem a descriure com s'han definit els elements clau mitjançant el sistema Lean i, simultàniament, realitzarem les demostracions de la Secció 2 de manera abreujada, comparant així la teoria amb les demostracions realitzades en Lean.

3.1. Axioma d'Elecció. Sota aquest nou paradigma comencem amb la declaració de definicions que utilitzarem per a donar pas a la definició de l'Axioma d'Elecció.

3.1.1. *Funció d'elecció.* A Lean, direm que un conjunt és una funció d'elecció si compleix les propietats que corresponen a la definició donada a l'Axioma d'Elecció de funció d'elecció (3.2).

```
def ChoiceFunction (f:Set.{u}) : Prop:=
∀(z w:Set.{u}), pair z w ∈ f ⟶ w ∈ z
```

3.1.2. *Axioma d'Elecció (AC).* Ja estem en condicions de definir l'Axioma d'Elecció segons l'Axioma 3.2 en Lean de la següent manera:

```
def AxiomOfChoice : Prop := ∀(𝒳:Set.{u}),
((∅:Set) ∉ 𝒳 ) ∧ Disj 𝒳  ⟶
∃f∈ 𝒳.funs 𝒳.sUnion, ChoiceFunction f
```

3.1.3. *Axioma d'Elecció sense la propietat de ser disjunt.* Definim l'Axioma d'Elecció sense la necessitat de que el conunt $\mathcal{X}$ siga disjunt com:

```
def AxiomOfChoice_noDisj :Prop:=∀(𝒳:Set.{u}),
((∅:Set) ∈ 𝒳 ) ⟶
∃f∈ 𝒳.funs 𝒳.sUnion, ChoiceFunction f
```

```
theorem AC_equiv_nodisj:
AxiomOfChoice.{u} ↔ AxiomOfChoice_noDisj.{u}
```

Suposem que es compleix l'Axioma d'Elecció `(AC)` (3.2), aleshores considerem $\mathcal{X}$ és un conjunt que no conté al buit `(E_notin_`$\mathcal{X}$ `: ∅ ∉` $\mathcal{X}$`)`.

```
intros AC 𝒳 E_notin_𝒳,
```

Definim el conjunt $\coprod \mathcal{X}$ que conté tots els conjunts amb etiqueta de $\mathcal{X}$.

```
set ∐𝒳:Set.{u}:=
{XX∈ powerset(prod (𝒳.sUnion) 𝒳)|∃X∈ 𝒳, XX = prod X {X
    }},
```

Mostrem que $\coprod \mathcal{X}$ compleix les propietats necessàries per a aplicar l'Axioma d'Elecció.

```
have hpropAC:(∅:Set.{u})∉∐𝒳 ∧ Disj ∐𝒳,
```

Primer demostrem que el buit no està contingut en $\coprod \mathcal{X}$.

```
--suposem que el buit està contingut en ∐𝒳. (hE:∅∈∐𝒳)
intro hE,
--si el buit està en ∐𝒳, aleshores existeix un conjunt X
    tal que (hX: X∈𝒳) i (E_eq_XX: ∅ = X×{X})
have E_in_XX,from mem_sep.mp hE,rcases E_in_XX.right with
    ⟨ X, hX, E_eq_XX ⟩,
--com X està en 𝒳, aleshores no és buit i per tant, conté
    un element x, tal que (hx:x ∈ X)
have Xne:Set_nonempty X, from xne_of_XhasNoE_of_xinX
    E_notin_𝒳 hX, cases Xne with x hx,
--com x∈X,  (x,X)∈ XX
have xX_in_XX:pair x X ∈ prod X ({X}:Set.{u}), [...]
--aleshores, com XX=∅, (x,X)∈ ∅
have xX_in_E: pair x X ∈ (∅:Set.{u}), from
    mem_of_eq_of_set E_eq_XX.symm xX_in_XX,
--el qual és una contradicció perque el buit no té
    elements
exact not_mem_empty (pair x X) xX_in_E,
```

En segon lloc, demostrem que $\coprod \mathcal{X}$ és disjunt.

Siguen $(X, \{X\})$ i $(Y, \{Y\})$ elements de $\coprod \mathcal{X}$, suposem que no són són disjunts, aleshores existeix un element $(a, b) \in (X, \{X\} \cap (Y, \{Y\}))$ de manera que obtenim que `(hbX:b∈{X})` i `(hbY:b∈{Y})` i per tant arribem a que $X \times \{X\} = Y \times \{Y\}$.

```
[...]--aleshores b∈ {X} ↔ b=X i b∈ {Y} ↔ b=Y
have b_eq_X:b=X,from mem_singleton.mp hbX,have b_eq_Y:b=Y
    , from mem_singleton.mp hbY,
--per tant X=b=Y ⟶  X × {X} = Y × {Y}
```

```
have X_eq_Y:X=Y,from eq.trans b_eq_X.symm b_eq_Y,
--definim l'aplicació que defineix un conjunt amb
    etiqueta.
set T:Set.{u}⟶Set.{u}:= λ X, X.prod X
--aleshores vegem que (X,{X}) = (Y,{Y})
have XX_eq_YY:X.prod {X} = Y.prod {Y}, from congr_arg T
    X_eq_Y,
[...]--així, queda demostrat Disj(∐𝒳)
```

Així doncs, en verificar que les condicions necessàries es compleixen, podem aplicar l'Axioma d'Elecció (AC) i obtenir una funció d'elecció

$$F : \coprod \mathcal{X} \longrightarrow \bigcup \left(\coprod \mathcal{X} \right). \tag{43}$$

```
specialize AC ∐𝒳 hpropAC, rcases AC with ⟨F, hF0,cfF⟩,
```

On `(hF0:f ∈ ∐𝒳.funs ∐𝒳.sUnion)` i `(cfF: ChoiceFunction F)`.

Aleshores podem definir una funció f com a la composició entre la funció F que hem obtés mitjançant (AC) i la funció que fa correspondre a cada element de $\mathcal{X}$ el seu conjunt amb etiqueta:

```
set TS:Set.{u}:={Z∈ 𝒳.prod ∐𝒳|∃X:Set.{u}, X.pair (X.
    prod {X}) = Z},
--vegem que està ben definit
have TSfunc:TS∈ 𝒳.funs ∐𝒳,[...]
--aleshores definim la composició
set f:Set.{u}:= ((mem_funs.mpr hF) ∘∘ TSfunc),
--Com que és composició de funcions, està ben definit
have hf:f∈ 𝒳.funs ∐𝒳.sUnion, from comp_is_func hF0
    TSfunc,
```

A continuació definim la funció π_1 que mapeja cada element d'un parell ordenat a la seua primera component.

```
set π₁:Set.{u}:={Z∈ ∐𝒳.sUnion.prod 𝒳.sUnion|
∃ X Y: Set.{u}, (X.pair Y).pair X = Z },
```

Demostrem que π_1 també està ben definida.

```
have hπ₁:π₁∈ ∐𝒳.sUnion.funs 𝒳.sUnion, [...]
```

Aleshores, definim la funció g com la composició de f i π_1 .

```
set g: Set.{u} := hπ₁ ∘∘ hf,
```

Mostrem que g està ben definida, a partir del fet de que la composició de funcions ho està.

```
have hg: g∈ 𝒳.funs 𝒳.sUnion, from comp_is_func hπ₁ hf,
```

Finalment demostrem que g és una funció d'elecció.

```
have hgCF: ChoiceFunction g, [...]
```

Aleshores, amb la demostració de que g és funció d'elecció, queda demostrat que l'Axioma d'Elecció no depèn de si el conjunt de partida és disjunt o no.

```
exact exists.intro g ((exists.intro hg) hgCF),
```

3.2. Tota funció sobrejectiva té inversa per la dreta. En aquesta secció definirem els conceptes clau per a realitzar l'equivalència 3.7 en Lean.

Per a resoldre aquesta equivalència primer anem a definir els següents conceptes que hem necessitat per a realitzar la equivalència en Lean.

3.2.1. *Funció sobrejectiva.* En primer lloc, la propietat sobrejectiva d'una funció $f : X \longrightarrow Y$ implica que, a cada element del codomini $y \in Y$ existeix un element $x \in X$ tal que $f(x) = y$, açò en Lean es defineix de la següent manera:

```
def is_surjective (hf: f∈ X.funs Y): Prop :=
∀y∈Y, ∃x:Set.{u}, x.pair y ∈ f
```

3.2.2. *Antiimatge.* Definim l'antiimatge d'un conjunt Z per una funció f com al conjunt que les imatges de tots els seus elements estan en Z. En lean s'expressa així.

```
def antiimage (hf: f∈ X.funs Y) (Z:Set.{u}):Set.{u} :=
{x∈ X| ∃y:Set.{u},y∈ Z ∧ x.pair y ∈ f}
notation hf⁻¹ Z:= antiimage hf Z
```

3.2.3. *Funció Identitat.* Definim la indentitat id_A com la funció que que a cada element de A li fa correspondre ell mateix. En Lean es defineix de la següent manera

```
def Id: Set.{u}:={x∈ A.prod A| ∃y:Set.{u}, y.pair y = x}
```

3.2.4. *Inversa per la dreta.* Diem que una funció g és la inversa per la dreta de f si $f \circ g = \mathrm{id}$, de manera que es defineix en lean com

```
def is_right_inverse
(hf: f∈ X.funs Y) (hg: g∈ Y.funs X):Prop:= hf ∘∘ hg = Id
    Y
```

Aleshores podem definir la existència d'una inversa per la dreta a una funció de la manera següent:

```
def has_right_inverse (hf: f∈ X.funs Y):Prop:=
∃(g:Set.{u})[hg: g∈ Y.funs X], is_right_inverse hf hg
```

Ara ja estem en condicions de demostrar el Teorema 3.7.

```
theorem AC_equiv_sur_to_RInv:
AxiomOfChoice.{u} ↔ ∀{X Y f:Set.{u}}(hf:f∈ X.funs Y),
is_surjective hf ⟶ has_right_inverse hf
```

Comencem demostrant que (AC) implica que tota funció sobrejectiva té inversa per la dreta.

Per a aconseguir-ho, primerament, considerem que es compleix l'Axioma d'Elecció (AC) i considerem $f : X \longrightarrow Y$ una funció sobrejectiva.

```
intros AC X Y f hf fSurj,
```

On `(hf)` fa referència a la propietat de ser f una funció entre els conjunts `X` i `Y`, mentre que `fSurj` és la hipòtesi que determina que f és una funció sobrejectiva.

Aleshores, definim $\mathcal{X}$ com al conjunt de totes les antiimatges d'un element de `Y`.

```
set 𝒳:Set.{u}:=
{Z ∈ powerset X | ∃ y:Set.{u}, y∈Y ∧  Z =  hf⁻¹ {y} },
```

Volem aplicar l'Axioma d'Elecció a aquest conjunt $\mathcal{X}$. Aleshores necessitem demostrar que $\mathcal{X}$ no conté al buit.

```
have E_notin_𝒳:(∅:Set.{u})∉ 𝒳,
```

Per a aconseguir el resultat, suposem per contradicció que $\emptyset$ està contingut en $\mathcal{X}$.

```
assume E_in_𝒳,
```

Com hem suposat `(E_in_`$\mathcal{X}$ `: ∅∈ `$\mathcal{X}$`)`, tenim que existeix un conjunt `(y:Set.{u})` tal que, `(hy:y ∈ Y ∧ ∅ = hf`$^{-1}${`y})`.

```
have hE𝒳,from (mem_sep.mp E_in_𝒳).right,
cases hE𝒳 with y hy,
```

Ara, com f és sobrejectiva tenim que existeix un element x en la preimatge de $\{y\}$ sobre f.

```
specialize fSurj y hy.left, cases fSurj with x hx,
have finv_ne:x∈ hf⁻¹{y}, [...]
```

Aleshores, a partir de `(finv_ne:`$x \in f^{-1}\{y\}$`)` i donat que `(hy.right.symm: hf`$^{-1}${`y}=∅)`, obtenim que x és un element del conjunt buit, i per tant hem arribat a una contradicció.

```
exact not_mem_empty x (mem_of_eq_of_set hy.right.symm
    finv_ne),
```

Ara podem aplicar l'Axioma d'Elecció obtenint la funció d'elecció:

$$F : \mathcal{X} \longrightarrow \bigcup \mathcal{X}$$

```
have AC_nodisj, from AC_equiv_nodisj.mp AC 𝒳 E_notin_𝒳,
rcases AC_nodisj with ⟨F, hF, CFF⟩,
```

A més podem definir la funció que a cada element `y` de `Y`, li fa correspondre l'antiimage de y:

$$G(y) = f^{-1}\{y\}.$$

```
set G:Set.{u}:=
{Z∈ Y.prod 𝒳 |∃y:Set.{u}, Z = (y.pair hf⁻¹{y})},
```

Aleshores, demostrem que aquesta funció està ben definida:

```
have hG:G∈ Y.funs 𝒳, [...]
```

A més, com $\bigcup \mathcal{X} \subseteq X$ obtenim que $F : \mathcal{X} \longrightarrow X$.

```
have hF2:F∈ 𝒳.funs X, [...]
```

De manera que ja estem en condicions per demostrar que $g = F \circ G$ és inversa per la dreta de f, en ser F funció d'elecció.

```
set g:= hF2 ∘∘ hG,
have hg: g∈ Y.funs X, from comp_is_func hF2 hG,
have RIfg:is_right_inverse hf hg, [...]
```

Com F és funció d'elecció, obtenim $f \circ g = \mathrm{id}$ i queda demostrada la primera implicació.

```
exact exists.intro g (exists.intro hg RIfg),
```

Comencem amb la implicació contrària.

Si tenim que tota funció sobrejectiva té inversa per la dreta, volem demostrar que es compleix l'Axioma d'Elecció.

Suposem que $\mathcal{X}$ és un conjunt disjunt que no conté al buit.

```
intros h_sur_to_Rinv 𝒳 𝒳.props,
```

Volem definir una funció $G : \bigcup \mathcal{X} \longrightarrow \mathcal{X}$ tal que a cada x de $\bigcup \mathcal{X}$, li fa correspondre $\bigcup\{X \in \mathcal{X} : x \in X\}$.

```
set G: Set.{u} :=
{z∈ 𝒳.sUnion.prod 𝒳| ∃ x X:Set, x ∈ X ∧ z=x.pair X},
have hG:G∈ 𝒳.sUnion.funs 𝒳, [...]
```

Ara demostrem que G és sobrejectiva.

```
have hGsur:is_surjective hG, [...]
```

Aleshores, com, per hipòtesi, tota funció sobrejectiva té inversa per la dreta, obtenim que existeix una inversa per la dreta de F tal que $G \circ F = \mathrm{id}$.

```
have GRinv, from h_sur_to_Rinv hG hGsur,
rcases GRinv with ⟨ F, hF, hFRinv ⟩,
```

A partir d'aquesta proposició obtenim que F és una funció d'elecció i amb açò queda demostrada la segon implicació i per tant l'equivalència.

```
have CFF:ChoiceFunction F, [...]
exact exists.intro F (exists.intro hF CFF),
```

3.3. Lema de Zorn-Kuratowski. L'equivalència del Lema de Zorn-Kuratowski té diversos punts amb els que difereix de la resta de les demostracions. Açò és degut, principalment, a que la ferramenta de formalització Lean, (com tantes altres) es basa en la teoria de tipus relacional, i no en la teoria de conjunts, que és mitjançant la qual hem realitzat la nostra demostració.

Dins de Lean, al tipus `Set` podem considerar el seu subtipus com als elements del tipus `Set` que compleixen una certa condició. Es per això que nosaltres utilitzem els següents tipus:

```
def to_set (P : Set.{u}) : set Set.{u} := {x | x ∈ P}
```

De manera que si x és del tipus `P.to_set` obtenim que `x.val` és de tipus `Set` i la proposició `(x.prop:x∈ P)`. No obstant, per a aconseguir definir un element de tipus `P.to_set` requerirem fer ús de la comanda `subtype.mk`, que, a partir d'un element `x` de tipus `Set` i una proposició `(h:x∈P)`, obtenim un element de tipus `P.to_set`. Per simplificar notació, en aquest cas definim:

```
def π {P:Set.{u}}(x : Set.{u})(h1 : x ∈ P):P.to_set :=
    subtype.mk x h1
notation ⟨x,h⟩:= π x h
```

Aquesta aplicació, a un conjunt `(x:Set.{u})` i una proposició `(h:x∈P)`, `⟨x,h⟩` és de tipus `P.to_set`.

De forma anàlega, siga `P` un conjunt, podem obtindre el subtipus dels subconjunts de `P` com al subtipus de les parts de `P`.

```
(powerset P).to_set : Type
```

En aquest cas definim la construcció del subtipus de la següent manera.

```
def phi {P:Set.{u}}(X : Set.{u}) (h1 : X ⊆ P) : (
    powerset P).to_set := subtype.mk X (mem_powerset.mpr
    h1)
```

Amb aquestes declaracions prèvies comencem amb la definició d'ordre parcial que ve donada en la biblioteca de Lean.

3.3.1. *Ordre parcial.* L'ordre parcial sobre un tipus α està definida a Lean com

```
class partial_order (α: Type u): Type u ⟶ Type u
(le: α ⟶ α ⟶ Prop)
(le_refl: ∀ (a:α), a ≤ a)
(le_trans: ∀ (a b c:α), a ≤ b ⟶ b ≤ c ⟶ a ≤ c)
(lt := λ a b, a ≤ b ∧ ¬ b ≤ a)
(lt_iff_le_not_le : ∀ a b : α, a < b ↔ (a ≤ b ∧ ¬ b ≤ a
    ) . order_laws_tac)
(le_antisymm:∀ (a b:α), a ≤ b ⟶ b ≤ a ⟶ a = b)
```

Podem observar que aquesta classe es equivalent a la Definició 3.9 de la Secció 2, però definida per a tipus. Direm que un tipus α té definit un ordre parcial si té una relació `le` o $\leq$ definida en α, que compleix la propietat reflexiva, transitiva i antisimètrica. A més, en aquesta definició incorpora l'ordre estricte com a part de l'ordre parcial, igual que a la Definició (3.10).

A partir d'aquest punt suposem `(P.to_set,≤)` un tipus parcialment ordenat.

3.3.2. *Cadena.* Per a definir una cadena `C` en Lean ho fem seguint la Definició 3.11.

```
def is_chain (C:(powerset P).to_set):Prop:= ∀ {x y : (P.
    to_set)},
x.1∈C.1 ⟶ y.1∈C.1 ⟶ x ≤ y ∨ y ≤ x
```

En aquest cas diem que, si `C` és de tipus `(powerset P).to_set`, és a dir, subconjunt de P, tot element `x,y` de tipus `P.to_set`, tal que `x.val∈C.val` i `y.val∈C.val`, compleix que `x≤y` o `y≤x`. Així mateix definim el conjunt de les cadenes de tipus `P.to_set`.

```
def chains :Set.{u}:= {C ∈ powerset P|∃(h1:C⊆P),
    is_chain (phi C h1)}
```

3.3.3. *Fita superior.* Definim la propietat de fita superior seguint la Definició 3.12.

```
def bounded (s :P.to_set)(X:(powerset P).to_set):Prop:=∀(
    x:Set.{u})(h1:x ∈ X.1), ⟨ x, ((mem_powerset.mp X.2) h1
    )⟩  ≤ s
```

3.3.4. *Suprem.* Per a definir l'element suprem en Lean ho fem seguint la Definició 3.13.

```
def suprem (s :P.to_set)(X:(powerset P).to_set):Prop:=
(bounded s X) ∧ (∀(x:Set.{u})(h1:x ∈ X.1), ⟨x, (X.2 h1)⟩
     ≤ s)
```

En aquest cas tenim que, si `s` és de tipus `P.to_set` i `X` de tipus `(powerset P).to_set`, aleshores tot element de `X.val`, és anterior o igual a `s`. Això es pot donar perquè tot element de `X.val` també ho és de `P` per `X.prop`.

3.3.5. *Element maximal.* Definim element superior seguint la Definició 3.14.

```
def element_maximal (M:P.to_set) :Prop:=
∀x:P.to_set , M ≤ x ⟶ M = x
```

En aquest cas obtenim que `M` de tipus `P.to_set` és maximal si no existeix cap element de tipus `P.to_Set` posterior a ell.

3.3.6. *Mínim.* Definim la propietat de ser un mínim d'un conjunt a partir de la Definició 3.15.

```
def minim_of_set (A:(powerset P).to_set)(m:P.to_set)(hm:m
    .1 ∈ A.1):=
∀(x:Set.{u})(hx:x ∈ A.1), m ≤ ⟨x, A.2 hx⟩
```

Ací, donats els elements `A` i `m` de tipus `(powerset P).to_set` i `P.to_set` respectivament, i la proposició que `m.val` ∈ `A.val`, direm que `m` és un mínim de `A` si, per a tot `X` element d'`A.val`, `m`≤ `x`.

3.3.7. *Segment inicial d'un conjunt fins a un element.* A Lean, hem definit segment inicial d'un conjunt fins a un element com a la Definició 3.17 de la següent manera.

```
def Cv (C: (powerset P).to_set){x:Set.{u}}(hx:x∈ C.1):Set
    .{u}:=
{y ∈ C.1 | ∃(hy:y∈C.1),⟨y,C.2 hy⟩ < ⟨x,C.2 hx⟩}
notation C`↓`hx:=Cv C hx
```

En aquest cas, siga `C` de tipus `(powerset P).to_set`, i `x` un element de `C.val`, definim `C`↓`x` com el conjunt que conté a tots els elements de `C` inferiors a `x`.

3.3.8. *Fites superiors estrictes d'un conjunt.* Definim el conjunt de les fites superiors estrictes d'un conjunt seguint la Definició 3.18.

```
def Upp (C: (powerset P).to_set) : Set.{u}:=
{x∈ P| ∃(hx:x∈P),∀(hy:y∈C.1), ⟨y, C.2 hy⟩  < ⟨x,hx⟩}
```

Donat `C` de tipus `(powerset P).to_set`. Definim `Upp(C)` com el conjunt amb els elements de `P` que són posteriors a tots els elements de `C`.

Lema 4.1 (Upp d'un conjunt buit és el total). *Donat un conjunt* `C` *buit subconjunt d'un tipus parcialment ordenat* `(P.to_set,≤)`*, obtenim que* `Upp(C) = P`.

```
lemma upp_empty_eq_P {C:(powerset P).to_set}(h1:C.val=(∅:
    Set.{u})): Upp(C) = P :=
```

Demostració (Lema 4.1). En primer lloc, per definició d'`Upp` obtenim que `Upp(C)`$\subseteq$`P`.

```
ext w, split, intro hw, exact (mem_sep.mp hw).left,
```

Considerem `hw:w`$\in$ `P`.

```
intro hw, apply mem_sep.mpr,split, exact hw,apply exists.
    intro hw,
```

Volem demostrar que $\forall$ `(y : Set) (hy : y` $\in$ `C.val)`, $\langle$`y,_`$\rangle <$ $\langle$`w,hw`$\rangle$. Per tant, si introduim la hipòtesi `(hy : y` $\in$ `C.val)`, com que `C.val` és buit, arribem a una contradicció, ja que el buit no pot tindre elements.

```
intros y hy, exact false.elim (not_mem_empty y (
    mem_of_eq_of_set h1 hy)),
```

Així queda demostrat el lema. □

3.3.9. *Lema de Zorn-Kuratwoski.* El Lema de Zorn-Kuratowski dictamina que, donat un conjunt no buit parcialment ordenat tal que tota cadena té suprem, aleshores existeix un element maximal d'eixe conjunt.

```
def preZorn (P:Set.{u})(hpo:partial_order (P.to_set)):
    Prop:=
(Set_nonempty P) ∧
(∀ (C:(powerset P).to_set),(C.1 ∈ chains P) ⟶ (
    Set_nonempty C.1) ⟶
(∃ s :P.to_set , suprem s C))
⟶ ∃ (M : P.to_set), element_maximal M

--Lema de Zorn--
def Zorn:Prop:=
∀(P:Set.{u})(hpo:partial_order (P.to_set)),preZorn P hpo
```

Com es pot observar hem definit el Lema de Zorn-Kuratowski en dues parts, això es deu a que, per com funciona Lean, per a poder utilitzar proposicions que utilitzen estructures com [`partial_order P.to_set`], aquestes necessiten estar definides abans de la definició. Es per això que, en aquest cas, com el Lema de Zorn-Kuratowski serveix per a tot conjunt parcial ordenat hem necessitat definir, primer, el Lema de Zorn-Kuratowski per a un conjunt específic, i després que, per a tot conjunt parcialment ordenat, es compleix.

```
theorem AC_iff_Zorn:AxiomOfChoice.{u} ↔ Zorn.{u}
```

Comencem amb la demostració de que, si es compleix l'Axioma d'Elecció, s'ha de complir el Lema de Zorn-Kuratowski.

Considerem que es compleix l'Axioma d'Elecció i suposem `P` un conjunt qualsevol no buit amb un ordre parcial definit i que tota cadena no buida de `P` té suprem.

```
intros AC P hpo hprops,
```

Suposem que no hi ha element maximal.

```
by_contra hnon_max,
```

3.3.11. *Lema A.* Si tenim que `P` no és buit, tota cadena no buida de `P` té suprem i no hi ha cap element maximal a `P`, volem demostrar que tota cadena `C` de tipus `(powerset P).to_set`, compleix que `Upp(C)` no és buit.

```
lemma LA
(Pne:Set_nonempty P)
(PWO:∀ (C : ((powerset P).to_set)), C.val ∈ chains P ⟶
   Set_nonempty C.val ⟶ (∃ (s : (P.to_set)), s
   suprem_de C))
(hnon_max:¬∃ (M : P.to_set), element_maximal M):
∀(C:(powerset P).to_set)(Cchain:is_chain C), (
   Set_nonempty (Upp C)):=
```

Demostració. Siga `C` una cadena de tipus `(powerset C).to_set`, suposem que `C.val=∅`.

```
intros C Cchain, by_cases hCne:C.1=(∅:Set.{u}),
```

Aleshores, com `P` no és buit existeix un element `x` de `P`.

```
cases Pne with x hx,
```

Obtenim que `x` ∈ `Upp(C)` mitjançant el Lema (4.1), demostrant així que no és buit.

```
have x_in_UppC:x∈ Upp(C), exact mem_of_eq_of_set (
   upp_empty_eq_P hCne).symm hx,
split, exact x_in_UppC,
```

Ara suposem que `C.val` no és buit. Aleshores, per la hipòtesi `PWO`, `C` té suprem, que anomenem `s`.

```
have supC, from PWO C C1chain Cne, cases  supC with s
   sSupC,
```

A continuació, obtenim que `s` no és maximal i per tant existeix un `x` posterior a `s`. `(xmax : s < x)`.

```
by_cases hmax:element_maximal s, exact false.elim (
   hnon_max (exists.intro s hmax)),
have xmax:∃x:P.to_set, s < x,from not_maximal_to_lt hmax,
    cases xmax with x xmax,
```

Finalment, tenim que `s` és suprem de `C`. Per tant, per a tot `y` de `C` `y≤s`. Aleshores, com hem demostrat que `s<x`, donat un `y` de `C`, `y<x` i, com a conseqüència, com `x.val` és un element de `C`, obtenim que `x.val∈Upp(C)`.

```
have s_lt_x:s<⟨x.1,x.2⟩,from eq.trans_gt pieq.symm xmax,
have x_in_UppC:x.val∈ Upp(C),split, exact x.prop, split,
   intros y hy, exact lt_of_le_of_lt (sSupC.left y hy) (
   s_lt_x),
exact exists.intro x.val x_in_UppC,
```

Així hem obtés que `Upp(C)` no és buit i queda demostrat el Lema A. □

Aleshores, definim el conjunt $\mathcal{C}$ com als conjunts superiors de totes les cadenes de `P`.

```
def 𝒞 :Set.{u}:={Z∈ powerset Q| ∃C:((powerset Q).to_set),
    is_chain C ∧ Z=Upp C}
```

Així, pel Lema A, obtenim que $\mathcal{C}$ no conté al buit.

```
lemma C_hasnt_E
(LemaA:∀(C:(powerset P).to_set)(Cchain:is_chain C), (
   Set_nonempty (Upp C))) :
(∅:Set.{u}) ∉ (𝒞 P) :=[...]
```

Per tant, podem aplicar l'Axioma d'Elecció no disjunt (3.1.3) sobre $\mathcal{C}$ i obtenim una funció d'elecció `F`.

```
specialize AC_nd (𝒞 P) (C_hasnt_E (LA hprops.left hprops.
   right hnon_max)),rcases AC_nd with ⟨F, Ffunc, CFF ⟩,
```

Aleshores definim un conjunt conforme com a la Definició 3.21.

```
class conforme (f:Set.{u}) (A:(powerset P).to_set):Prop:=
(C0:Set_nonempty A.1)
(C1:∀(X:Set.{u})(hss:X⊆A.1), Set_nonempty X ⟶ (∃(x:P.
   to_set)(hx:x.1∈X),minim_of_set (phi X (previs.
   subset_trans hss A.2)) x hx))
(C2:∀(x:Set.{u})(hx: x∈ A.1), (Upp (phi (A↓hx) (Cv_ss_P))
   ).pair x ∈ f)
```

I a més, demostrem que tot conjunt conforme és cadena.

```
lemma confCad {f:Set.{u}} {A:(powerset P).to_set}:
   conforme f A ⟶ is_chain A:=[...]
```

Aleshores estem en condicions de demostrar el Lema B.

3.3.12. *Lema B.* Si `A` i `B` de tipus `(powerset P).to_set`, són conformes i `A\B` és no buit, aleshores existeix un element `x`, mínim d'`A\B`, tal que `B.val=A↓x`.

```
lemma LB {f:Set.{u}} {A B:(powerset P).to_set} (fFunc:f∈
    (𝒞 P).funs (sUnion 𝒞 P) ) (hA:conforme f A)(hB:
    conforme f B):
Set_nonempty (A.1\B.1)⟶
(∃(x:Set.{u})(h1:x∈ (A.1\B.1)), B.1 = (A↓(mem_diff.mp h1)
    .left) ∧ minim_of_set (phi (A.1\B.1) _) ⟨x,_⟩ h1 ):=
```

Demostració. En primer lloc, si `A\B` no és buit, aleshores, té mínim `x`, volem demostrar que `B.val=A↓x`. A banda, per simplificar, denotem `Ax` a `A↓x`.

```
intros AlBne,
--com que A\B ⊆ A
have AlB_ss_A: A.1\B.1 ⊆ A.1, exact dif_subset,
--Aleshores tenim que A.1\B.1 té mínim
have AlBmin:∃(x:P.to_set)(hx:x.1∈(A\B)),minim_of_set (phi
    (A\B) (previs.subset_trans (dif_subset A.2))) x hx))
    ,from [...], rcases AlBmin with ⟨x, hx, xmin⟩,
--Definim Ax:= A↓x
set Ax:= (A↓(mem_diff.mp hx).left),
-- només cal demostrar la següent igualtat
have LemaB: B.val = Ax,
```

Per a aconseguir-ho, primer demostrem que `A↓x ⊆ B.val`.

Suposem que `w∈A↓x` però no està en `B.val`, aleshores arribem a que `x≤w<x`.

```
intro w_in_Ax, by_contra w_notin_B,
have x_irrefl:x<x,[...],
exact lt_irrefl x x_irrefl,
```

Per acabar el Lema B demostrem que `B.val⊆ A↓x`.

Suposem que `w∈B.val` però no està en `A↓x`,

```
intros hw, by_contra w_notin_Ax,
```

Aleshores, `B.val\A↓x` és un subconjunt de `B.val` no buit, per tant té mínim `y`. Denotem per `By=B↓y`

```
have BlAxne:Set_nonempty (B.val\Ax),[...],
have BlAxss:(B.val\Ax)⊆ B.1, from dif_subset,
have BlAx_min, from CC1 hB BlAxss BlAxne, rcases BlAx_min
    with ⟨y,hy,ymin⟩,
set By:Set.{u}:=(B↓(mem_diff.mp hy).left),
```

D'altra banda, demostrem el Lema B1.

```
have LB1: ∀{v u:P.to_set}(h1 :v.1 ∈ A.val)(h2:u.1∈ By), v
    < u ⟶ v.1∈ By, [...]
```

Com a conseqüència de la hipòtesi de que `A.val\B.val` no és buit, obtenim que `A.val\By` és un subconjunt no buit d'`A.val` i aleshores, per ser `A` conforme, obtenim que existeix un mínim `z`.

```
have AlBy_ne:Set_nonempty (A.val\By),[...]
have AlBy_ss_A:(A.val\By)⊆ A, from dif_subset,
have hzmin, from CC1 hA AlBy_ss_A AlBy_ne, rcases hzmin
   with ⟨z,hz,zmin⟩,
set Az:Set.{u}:=(A↓(mem_diff.mp hz).left),
```

Aleshores demostrem el Lema B2:

```
have LB2:Az=By,[...]
```

En conclusió, per la segona propietat de conjunts conformes, obtenim que `z=y`.

```
have z_eq_y1:z=y,[...]
```

Finalment, arribem a que `y≤ x` i per tant, utilitzant la funció `lt_or_eq_of_le` obtenim que, o bé `y<x` i arribem a que `x.val∈Ax` o bé `y=x` i arribem a que `x.val∈B.val` el qual també és una contradicció per definició de `x`.

```
have y_le_x: y ≤ x,[...]
have y_lt_x, from lt_or_eq_of_le y_le_x, cases y_lt_x,
--y < x--
    have y_in_Ax:y.1 ∈ Ax, [...],
    exact (mem_diff.mp hy).right y_in_Ax,
--y = x--
    have x_in_B:x.1∈ B.1, [...],
    have hx2:x.1∉B.1, from (mem_diff.mp hx).right,
    exact hx2 x_in_B,
```

Així, queda demostrat el Lema B. □

Aleshores estem en condicions suficients per demostrar el Lema C.

3.3.13. *Lema C.* El Lema C diu que, la unió d'una família no buida de conjunts conformes també és conforme.

```
lemma LC {𝒜 f:Set.{u}}
(fFunc:f∈ (𝒞 P).funs (sUnion 𝒞 P))
(A_ss_PP:𝒜 ⊆ powerset P )
(h𝒜: ∀{A:Set.{u}}(hA:A∈𝒜), conforme f (phi A (
   mem_powerset.mp (A_ss_PP hA))))
(hne:Set_nonempty 𝒜 ):
conforme f (phi (sUnion 𝒜) (sUnion_A_in_P A_ss_PP)):=
```

DEMOSTRACIÓ. Per a demostrar-ho, tenim que sUnion $\mathcal{A}$ no és buit, ja que existeix un A$\in \mathcal{A}$ i com A és conforme tampoc és buit.

```
cases hne with A hA, cases (h𝒜 hA).C0 with a ha,
split,exact mem_sUnion.mpr (exists.intro A (exists.intro
    hA ha)),
```

En segon lloc, demostrem que tot subconjunt no buit de sUnion $\mathcal{A}$ té mínim. Siga X un subconjunt no buit d'$\mathcal{A}$.

```
intros X hss Xne,
```

Com X no és buit existeix un element a de X$\subseteq$ sUnion $\mathcal{A}$, de manera que existeix un conjunt A de $\mathcal{A}$ tal que a$\in$ A.

```
cases Xne with a ha,
rcases (mem_sUnion.mp (hss ha)) with ⟨ A, hA, a_in_A⟩,
have hAconf,from h𝒜 hA,
```

Aleshores tenim que X$\cap$A és subconjunt d'A, que és conforme, i per tant té un mínim z.

```
have XcapAne:Set_nonempty (X∩A),split, exact mem_inter.
    mpr (and.intro ha a_in_A),
rcases (CC1 hAconf XcapA_ss_A XcapAne) with ⟨z,hz,zmin ⟩,
```

Ara bé, demostrem que eixe z és el mínim. Siga b un element de X.

```
split,split,
intros b hb,
```

Si b$\in$A, per tant z$\leq$b.

```
by_cases b_in_A:b∈ A, exact zmin b (mem_inter.mpr (and.
    intro hb b_in_A)),
```

Si b$\notin$A, obtenim que existeix un B$\in \mathcal{A}$ conforme tal que B\A és un subconjunt no buit de B.

```
rcases (mem_sUnion.mp (hss hb)) with ⟨B,hB,b_in_B⟩,
have hBconf, from h𝒜 hB,
have b_in_BlA,from mem_diff.mpr (and.intro b_in_B b_in_A)
    ,
have BlAne:Set_nonempty (B\A), split, exact b_in_BlA,
```

Aleshores, pel Lema B, obtenim que existeix un mínim x de B\A, tal que A=B$\downarrow$x.

```
rcases (LB fFunc hBconf hAconf BlAne) with ⟨x,hx,A_eq_Bx,
    xmin⟩,
```

Així doncs, com z$\in$ A=Bx, tenim que z<x$\leq$ b.

```
have x_le_b, from xmin b b_in_BlA,
have z_in_Bx, from mem_of_eq_of_set A_eq_Bx (mem_inter.mp
    hz).right,
have z_le_x, [...],
exact le_trans z_le_x x_le_b,
```

Ara només queda demostrar que per a tot `a` de `sUnion` $\mathcal{A}$, se satisfà que `f (Upp(sUnion` $\mathcal{A}$ `↓x)) = x`. Siga `a` un element de `sUnion` $\mathcal{A}$ existeix un `A` de $\mathcal{A}$ conforme que conté a `a`.

```
intros a ha, rcases (mem_sUnion.mp ha) with ⟨A,hA, a_in_A
    ⟩,
  have hAconf, from h𝒜 hA,
```

Tenim que `A` i `sUnion` $\mathcal{A}$ estan contingudes en `P`.

```
have A_ss_P:A⊆P, from mem_powerset.mp (A_ss_PP hA),
have UA_ss_P:sUnion 𝒜⊆P, intros x hx,rcases (mem_sUnion.
    mp hx) with ⟨ Y,hY,x_in_Y ⟩, exact mem_powerset.mp (
    A_ss_PP hY) x_in_Y,
```

Aleshores obtenim que `A↓ a =` `sUnion` $\mathcal{A}$ `↓a`.

```
have Aa_eq_U𝒜a:((phi A A_ss_P)↓a_in_A)=((phi (sUnion 𝒜)
    UA_ss_P)↓ha), [...]
```

Per tant, com `A` és conforme, es dona la següent igualtat

$$\texttt{F(Upp ((sUnion } \mathcal{A}\texttt{)↓a)=F(Upp A↓ a) = a.}$$

```
have UppAa_in_f: (Upp ( phi ((phi A A_ss_P)↓a_in_A)
    Cv_ss_P)).pair a ∈ f, from CC2 hAconf,
exact mem_of_eq_of_mem_left UppAa_eq_UppUAa UppAa_in_f,
```

Així doncs, queda demostrat el Lema C. □

Tot seguit, definim el conjunt de conjunts conformes $\mathcal{K}$.

```
def 𝒦 (f:Set.{u}):Set.{u}:={A∈ powerset Q| ∃(h1:A⊆ Q),
    conforme f (phi A h1) }
```

Ara, demostrem que $\mathcal{K}$ no és buit per a obtindre que `sUnion` $\mathcal{K}$ és conforme pel Lema C.

```
lemma U𝒦conf (fFunc:f∈ (𝒞 P).funs (sUnion 𝒞 P)):
    conforme f (phi (sUnion (𝒦 P f)) (sUnion_A_in_P (
    K_ss_P P f))):=
```

Aleshores, com tenim que tot conjunt conforme és cadena, i que `sUnion` $\mathcal{K}$ és conforme, obtenim que `sUnion` $\mathcal{K}$ està en $\mathcal{C}$ i per tant anomenem `k` a la imatge de `sUnion` $\mathcal{K}$ per `F`.

```
have UK_in_C:UppUK∈ 𝒞 P, [...]
rcases (mem_funs.mp Ffunc).right (UppUK) UK_in_C with ⟨k,
    hk,kuniq ⟩ ,
```

Tenim que k∈Upp(sUnion $\mathcal{K}$) i per tant $k \notin$sUnion $\mathcal{K}$.

```
have k_in_UppUK:k ∈ UppUK, from CFF UppUK k hk,
--per tant kUppUK := ∀x∈ sUnion 𝒦, x < k
cases (mem_sep.mp k_in_UppUK).right with k_in_P kUppUK,
--aleshores k∉ sUnion 𝒦
have k_notin_UK:k∉UK,
```

Aleshores demostrem que {k}∪sUnion $\mathcal{K}$ és conforme.

```
lemma kuUKconf (fFunc:f∈ (𝒞 P).funs (sUnion 𝒞 P)) {k:Set
   .{u}}(k_in_P:k∈ P) (hk:∀ (y : Set) (hy : y ∈ (sUnion
   (𝒦 P f))), ⟨ y,(sUnion_A_in_P (K_ss_P P f)) hy ⟩ < ⟨k,
   k_in_P⟩) (hkf:(Upp (phi (sUnion  𝒦 P f) (sUnion_A_in_P
    (K_ss_P P f)))).pair k ∈ f):
conforme f (phi ({k}∪(sUnion (𝒦 P f))) (kuUK_ss_P k_in_P
   )):=[...]
```

Així, arribem a que k∈sUnion $\mathcal{K}$, el que és una contradicció amb la proposició que haviem demostrat.

```
have kuUK_in_K: ({k}∪ UK)∈ 𝒦 P F, apply mem_sep.mpr,
   split, exact mem_powerset.mpr (kuUK_ss_P k_in_P),
   split, exact kuUKconf Ffunc k_in_P kUppUK hk,
have k_in_kuUK:k∈ ({k}∪ UK), from mem_union.mpr (or.inl (
   mem_singleton.mpr rfl)),
have k_in_UK:k∈ UK, from mem_sUnion.mpr (exists.intro ({k
   }∪ UK) (exists.intro kuUK_in_K k_in_kuUK)),
exact k_notin_UK k_in_UK,
```

Així doncs, queda demostrat que l'Axioma d'Elecció implica el Lema de Zorn-Kuratowski en Lean.

Comencem doncs, la demostració de que el Lema de Zorn-Kuratowski implica l'Axioma d'Elecció.

En primer lloc, suposem que es compleix el Lema de Zorn-Kuratowski i considerem un conjunt $\mathcal{X}$ que no conté al conjunt buit.

```
intros hzorn 𝒳 𝒳_props,
```

Definim el conjunt P que conté a les parelles de conjunts (𝒴,G) on $\mathcal{Y} \subseteq \mathcal{X}$, G∈ $\mathcal{Y}$. funs $\mathcal{Y}$.sUnion i G és funció d'elecció. Aquesta definició és la que obtenim a l'Equació 33.

```
def P_ (𝒳:Set.{u}) :Set.{u}:= {Z∈ (powerset 𝒳).prod (
   powerset (𝒳.prod 𝒳.sUnion))| ∃ 𝒴 G:Set.{u}, G∈(𝒴.
   funs 𝒴.sUnion) ∧ (∀{x y:Set.{u}}, x.pair y ∈ G ⟶ y∈
   x) ∧  Z=𝒴.pair G}
```

Aleshores definim el següent ordre parcial sobre aquest conjunt.

```
def hle_ (P:Set.{u}):(P.to_set) ⟶(P.to_set) ⟶ Prop:=λ
    a b, ∃(X Y G H:Set.{u}),
     (X ⊆ Y)
     ∧ (G ⊆ H)
     ∧ (X.pair G = a.1 ∧  Y.pair H = b.1)
```

Per a poder utilitzar un ordre parcial concret hem de definir una instància.

```
structure PartialOrderInstance :=
  (α : Type u)
  (le : α ⟶ α ⟶ Prop)
  (partial_order_proof : partial_order α)
```

A partir d'aquesta definició, ja podem, demostrant que és ordre parcial, utilitzar-lo més endavant.

Demostrem que `hle_` constitueix un ordre parcial. Per a fer-ho, demostrem que es compleixen les condicions per a certificar que `hle_` és un ordre parcial.

```
lemma hle_antisymm :∀ (a b : ((P_ 𝒳).to_set)), hle_ (P_
    𝒳) a b ⟶ hle_ (P_ 𝒳) b a ⟶ a = b:=[...]
lemma hle_refl:∀ (a : ((P_ 𝒳).to_set)),hle_ (P_ 𝒳) a a
    :=[...]
lemma hle_trans:∀ (a b c: ((P_ 𝒳).to_set)),hle_ (P_ 𝒳) a
     b ⟶ hle_ (P_ 𝒳) b c ⟶hle_ (P_ 𝒳) a c:=[...]
```

Aleshores obtenim que `hle_` constitueix un ordre parcial que podem utilitzar a la demostració.

```
def myPartialOrderInstance (𝒳:Set.{u}) :
    PartialOrderInstance :=
  {
    α := ((P_ 𝒳).to_set),
    le:= hle_ (P_ 𝒳),
    partial_order_proof:= {
      le:= hle_ (P_ 𝒳),
      le_antisymm:= hle_antisymm,
      le_refl:=hle_refl,
      le_trans:=hle_trans
    }
  }

  def hpo (𝒳:Set.{u}):=(myPartialOrderInstance 𝒳).
   partial_order_proof
```

Ara, suposem que $\mathcal{X}$ és el buit.

```
def hpo (𝒳:Set.{u}):=(myPartialOrderInstance 𝒳).
    partial_order_proof
```

Aleshores podem definir l'aplicació buida

```
set f:={x∈(∅:Set.{u}).prod (∅:Set.{u}).sUnion| false},
```

Aleshores, demostrem que aquesta funció està ben definida i és funció d'elecció i obtenim que es compleix l'Axioma d'Elecció.

```
have h1:f ∈ 𝒳.funs 𝒳.sUnion,[...]
have h2:ChoiceFunction f, intros z w hzw,
    exact false.elim (mem_sep.mp hzw).right,
exact exists.intro f (exists.intro h1 h2),
```

Així doncs, podem suposar que $\mathcal{X}$ no és buit, aleshores vegem que `P` no és buit.

```
lemma Pne (h𝒳E:𝒳≠ (∅:Set.{u}))(𝒳_props: ∅ ∉ 𝒳) : (P_ 𝒳)
    .nonempty:=
```

Com que $\mathcal{X}$ no és buit existeix un conjunt `X` contingut en ell, i al mateix temps `X` tampoc és buit. Per tant existeix un element `x` de `X`.

```
have 𝒳ne:Set_nonempty 𝒳, from nonempty_iff_notEmpty.mpr
    h𝒳E, cases 𝒳ne with X hX,
have Xne:Set_nonempty X, from xne_of_XhasNoE_of_xinX 𝒳
    _props hX, cases Xne with x hx,
```

Així doncs, podem definir la funció que, al conjunt `X` li fa correspondre `x`, o el que és el mateix, el conjunt que únicament conté a `(X,x)`.

```
set J:Set.{u}:={X.pair x},
```

Demostrem que està ben definit.

```
set X_:Set.{u}:={X},
have hJ:J∈ funs X_ X_.sUnion,
```

Ara, demostrem que compleix totes les condicions per a estar en `P`.

```
have XY_in_P1:X_.pair J ∈ (powerset 𝒳).prod (powerset (𝒳
    .prod 𝒳.sUnion)),apply pair_mem_prod.mpr,split,
--⊢ {X}∈ (powerset 𝒳)
    have h2: {X}⊆ 𝒳, intros w hw, exact mem_of_eq_of_mem
    (mem_singleton.mp hw).symm hX,
    exact mem_powerset.mpr h2,
--⊢{X.x} = J ⊆ (𝒳.prod 𝒳.sUnion)
    have Xx_in_XUX:X.pair x ∈ 𝒳.prod 𝒳.sUnion,[...]
    exact mem_powerset.mpr (λ w hw, (mem_of_eq_of_mem (
   mem_singleton.mp hw).symm Xx_in_XUX)),
--⊢ ∀ (x y : Set), x.pair y ∈ J ⟶ y ∈ x
```

```
9     have hcf:∀ (x y : Set), x.pair y ∈ J ⟶ y ∈ x,
    intros w y hwy,[...]
```

Aleshores tenim que ({$\mathcal{X}$},J)∈P.

```
1 have XJ_in_P: (X_.pair J)∈ P_ 𝒳, [...]
2 exact exists.intro (X_.pair J) XJ_in_P,
```

En segon lloc, demostrem que tota cadena no buida de P té suprem.

```
1 lemma PWO {𝒳:Set.{u}} : ∀ (C: (powerset (P_ 𝒳)).to_set )
    ,(C.1 ∈ (chains2 (P_ 𝒳) (hpo 𝒳)) ⟶ (Set_nonempty C
    .1)  ⟶ (∃ s : (P_ 𝒳).to_set , suprem2 (P_ 𝒳) (hpo 𝒳
    ) s C)):=
```

Considerem C una cadena no buida de P.

```
1 intros C C_chain C_ne,
```

Definim els següents conjunts:

```
1 set 𝒵:Set.{u}:= {Y∈𝒳 | ∃𝒴 G:Set.{u}, (𝒴.pair G ∈ C.1) ∧
    (Y ∈ 𝒴)},
2 set 𝒢:Set.{u}:= {Z∈ 𝒳.prod 𝒳.sUnion | ∃ X Y G 𝒴 :Set.{u
    }, (X∈ 𝒴) ∧ (X.pair Y ∈ G) ∧ (𝒳.pair G ∈ C.1)  ∧ Z=X.
    pair Y},
```

Corroborem que G està ben definida.

```
1 have G_fun_Z: 𝒢 ∈ 𝒵.funs 𝒵.sUnion,[...]
```

Comprovem que $\mathcal{G}$ és funció d'elecció.

```
1 have CFG:(∀x y:Set.{u}, x.pair y ∈ 𝒢 ⟶ y∈ x), [...]
```

Per últim demostrem que $\mathcal{Z}$.pair $\mathcal{G}$ estan en el conjunt que conté a P.

```
1 have ZG_in_PXPXUX:𝒵.pair 𝒢 ∈ (powerset 𝒳).prod (powerset
     (𝒳.prod 𝒳.sUnion)), [...]
```

Aleshores obtenim que $\mathcal{Z}$.pair $\mathcal{G}$ estan en P.

```
1     have ZG_in_P: 𝒵.pair 𝒢 ∈ (P_ 𝒳),
```

Així doncs, $\mathcal{Z}$.pair $\mathcal{G}$ són suprems de C.

```
1 have ZGsup: suprem2 (P_ 𝒳) (hpo 𝒳) ⟨𝒵.pair 𝒢 ,ZG_in_P⟩
    C,
2 split, exact ZGsup,
```

Per tant, podem aplicar el Lema de Zorn-Kuratowski i obtenim que existeix un element maximal M=(N,G) de P.

```
specialize hzorn (P_ 𝒳) (hpo 𝒳) (and.intro (Pne h𝒳E 𝒳
    _props) PWO),
cases hzorn with M M_max, cases M with M M_in_P,
rcases (mem_sep.mp M_in_P) with ⟨hMX,N,G,Gfunc,CFG,Meq⟩,
```

L'únic que queda per demostrar és que N=$\mathcal{X}$. Per a aconseguir-ho, suposem que són distints.

```
have N_eq_X:N=𝒳,
    by_contra N_neq_X,
```

Aleshores, com que N$\subseteq \mathcal{X}$ existeix algun element de $\mathcal{X}$\N. Considerem així, X com un element de $\mathcal{X}$\N, que com està contingut en $\mathcal{X}$ i aquest no conté al buit, existeix un element x en X.

```
have N_ss_X: N⊆𝒳, from mem_powerset.mp (
    pair_mem_prod_left (mem_of_eq_of_mem Meq hMX)),
have XlNne: Set_nonempty (𝒳\N), from
    diff_nonempty_of_subsetneq N_ss_X N_neq_X,
cases XlNne with X hX, have hX, from mem_diff.mp hX,
    cases hX with hX X_notin_N,
have Xne:Set_nonempty X, from xne_of_XhasNoE_of_xinX 𝒳
    _props hX, cases Xne with x hx,
```

Aleshores demostrem que la funció que definim a l'Equació 39 que es defineix a Lean com G∪{X.pair x} compleix que

(N∪{X},G∪{(X, x)})∈P.

```
have GuXx_in_P:(N∪{X}).pair (G∪{X.pair x})∈ P_ 𝒳,
```

Per a procedir amb la demostració, comprovem que (G∪{X.pair x}) està definida a (N∪{X}).funs (N∪{X}).sUnion.

```
lemma WDGA (fFunc:f∈ Y.funs Y.sUnion)(hx:x∈X)(X_notin_Y:X
    ∈Y): f∪{X.pair x}∈ (Y∪{X}).funs (Y∪{X}).sUnion:=[...]
```

Finalment demostrem la resta de propietats que s'han de complir per a aconseguir l'objectiu al que volíem arribar.

```
have CFGUX: ∀(a b:Set.{u}), a.pair b ∈ G∪{X.pair x} ⟶ b
    ∈ a,
have hfinale:(N∪{X}).pair (G∪{X.pair x})∈𝒳.powerset.prod
     (𝒳.prod 𝒳.sUnion).powerset,
```

Amb açò, hem demostrat que (N∪{X},G∪{(X, x)})∈P. En conseqüència, obtenim que

```
M≤(N∪{X},G∪{(X, x)})
```

```
have leDEF: (hle_ (P_ 𝒳)) ⟨M,M_in_P⟩ ⟨(N∪{X}).pair (G∪{X.
    pair x}), GuXx_in_P ⟩,
```

Però com `M` és maximal, implica que `M=(N∪{X},G∪{(X, x)})`, però aleshores tindriem que `N=N∪{X}` el qual, per hipòtesi, `X∉N`.

```
specialize M_max ⟨(N∪{X}).pair (G∪{X.pair x}), GuXx_in_P
    ⟩ leDEF,
have M_eq_NGuXx: M = ((N∪{X}).pair (G∪{X.pair x})), from
    congr_arg subtype.val M_max,
have N_eq_NuX:N=N∪{X}, from pair_inj_left (eq.trans Meq.
    symm M_eq_NGuXx),
have X_ss_N:{X}⊆ N, from subset_of_subset_of_union_left
    (previs.subset_of_eq N_eq_NuX.symm),
have X_in_N:X∈N, from mem_of_subsingleton_subset.mp
    X_ss_N,
exact X_notin_N X_in_N,
```

Aleshores, com `N` $=\mathcal{X}$, tenim que `G` és una funció d'elecció en `N`$=\mathcal{X}$. Aquesta és la funció d'elecció que buscàvem.

```
have h1:G ∈ 𝒳.funs (sUnion 𝒳), from N_eq_X ▷ Gfunc,
have h2:ChoiceFunction G, intros a b hab, exact CFG hab,
exact exists.intro G (exists.intro h1 h2),
```

Queda així concluida la demostració de la implicació i la equivalència.

CAPíTOL 5

Conclusió

En aquest capítol, concloem amb el treball final de grau, on hem explorat a fons la formalització de diverses equivalències vinculades a l'Axioma d'Elecció mitjançant el llenguatge de verificació formal Lean. Al llarg d'aquest procés, hem tingut l'oportunitat de submergir-nos en l'emocionant món de la verificació formal, un camp en constant evolució que promet canviar la forma en la que abordem i validem els conceptes matemàtics.

El nucli d'aquest treball s'ha enfocat en la teoria axiomàtica de conjunts, un pilar fonamental en la història de les matemàtiques que coneguem a dia de hui. En aquesta àrea, hem desenvolupat de manera integral totes les equivalències, sense necessitat de recórrer a demostracions externes. Aquest enfocament ha permés una comprensió profunda i rigorosa dels principis subjacents, resaltant el potencial de la verificació formal per a demostrar i validar teoremes de manera concloent.

Al llarg d'aquesta investigació, hem experimentat, de primera mà, com aquestes ferramentes de verificació formal, com és el cas de Lean, Coq o d'altres, estàn transformant la manera en la que es poden abordar les demostracions matemàtiques. El més emocionant és la perspectiva de futur, a mesura que la intel·ligència artificial segueix evolucionant, permeten fer que aquestes ferramentes siguen encara més intuitives i efectives. Aquesta perspectiva obre la porta a una nova era en la que la verificació de teoremes es torne més accessible i precisa, amb una reducció substancial d'errades humanes.

A més, és crucial recordar que aquest projecte es va originar a partir de la investigació sobre la ferramenta de verificació formal Lean. Cal destacar com l'ús de verificadors formals han tingut un paper crucial en la millora de la lògica en aquestes demostracions, enfatitzant la importància d'equilibrar la intuició humana amb una lògica rigorosa, ja que, si bé la intuició és essencial per a la creació de demostracions, ferramentes com aquesta proporcionen una garantia sòlida de coherència i validesa.

En resum, aquesta conclusió marca un final en aquest viatge de descobriment sobre la convergència entre les matemàtiques i la informàtica

que s'està vivint a dia de hui, senyalant el potencial transformador de les ferramentes de verificació formal. L'evolució sense fi d'aquestes ferramentes i el seu impacte prometedor en el camp de les matemàtiques són raons sòlides per sentir entusiasme pel futur d'aquesta disciplina. Amb aquesta investigació buscàvem també resaltar el paper fonamental que aquestes ferramentes poden tindre en les matemàtiques modernes i el prometedor futur que els espera.

A més, aspirem a que aquest treball també servisca com a guia per a aquells que desitgen endinsar-se en la formalització en la teoria de conjunts amb aquesta ferramenta.

Bibliografia

[ADMK17] J. Avigad, L. De Moura, and S. Kong. Dependent type theory. `https://lean-lang.org/theorem_proving_in_lean/dependent_type_theory.html`, 2017.

[Bar08] H. P. Barendregt. Structure and interpretation of computer programs. *The MIT Press*, 2008.

[BDS13] H. P. Barendregt, W. Dekkers, and R. Statman. Lambda calculus with types. *The Computer Journal*, june 2013.

[Buk19] B. Bukh. *Math Studies Algebra: Axiom of Choice.* Addison-Wesley Professional, 2 edition, 2019.

[CV10] J. Climent Vidal. *Teoría de conjuntos*. Disponible en linea, 2010.

[DM15] L. De Moura. The lean theorem prover (system description). *The Computer Journal*, january 2015.

[LC] Lean-Community. mathlib3 documentation. `https://leanprover-community.github.io/mathlib_docs/`. Biblioteca matemàtica de Lean3.

[Lew91] J. W. Lewin. A simple proof of zorn's lemma. *The American mathematical monthly*, 98(4):353–354, 1991.

[ML80] P. Martin-Löf. Intuitionistic type theory, 1980.

[Ope22] OpenAI. Solving (some) formal math olympiad problems. `https://openai.com/research/formal-math`, 2022.

[PL23] V. Pons Llopis. Repositori de les equivalencies en l'axioma d'eleccio en lean3. `https://github.com/vpons2000/LEAN3-Equivalencies_en_Axioma_Eleccio`, 2023.

[WR27] A. N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1925–1927.