Mathematical Foundations

David McAllester

Draft of December 12, 2012

Contents

Introduction, Overview and Historical Remarks

This book grew out of a course taught at the Toyota Technological Institute at Chicago designed to raise the level of mathematical maturity of students studying machine learning. The first half of the book develops Boolean type theory — a type-theoretic formal foundation for mathematics designed specifically for this course. Boolean type theory allows much of the content of mathematical maturity to be formally stated and proved as theorems about mathematics in general. The second half of the book is a series of case studies in particular mathematical concepts illustrating the general principles developed in the first half. The topics in these case studies have been selected so as to be relevant to machine learning and emphasize topics related to linear algebra.

Students are usually expected to attain maturity by taking introductory classes such as discrete mathematics, real analysis, or linear algebra. The content of mathematical maturity is somehow implicit in these courses. This book, on the other hand, directly formulates much of the content of mathematical maturity allowing this content to be explicitly taught.

A first hallmark of maturity is the ability to distinguish statements that are well formed from those that are not. For example, a statement involving a sum of a scalar and a vector is not well formed — we can add two vectors but we cannot add a vector and a scalar. The general notion of well-formedness fundamentally involves types. For example, the types "scalar" and "vector". A fundamental property of mathematics is its ontological richness — the richness of types occurring in mathematical discourse. One can talk about integers, reals, groups, rings, fields, vector spaces, Hilbert spaces, Banach spaces and so on. Well-formedness is formulated here in terms of a type system for mathematics.

A second hallmark of mathematical maturity is an intuitive understanding of the notion of isomorphism. Isomorphism is meaningful for every kind of mathematical object. We have isomorphic groups, graphs, fields, topological spaces, differentiable manifolds and so on. Every mathematical mature person knows, for example, that no well-formed statement about an abstract graph can distinguish between isomorphic graphs. The type theory developed here formulates a general principle of substitution of isomorphics — two isomorphic objects are substitutable for each other in type-theoretically well formed contexts.

The notion of isomorphism is related to the notion of an abstract interface in computer programming. An abstract interface specifies what information and behavior an object provides. Two different implementations can produce identical behavior as viewed through an abstract interface. An example of this is given by the real numbers. Textbooks on real analysis typically start from axioms involving multiplication, addition, and ordering. Addition, multiplication and ordering define an abstract interface — the well-formed statements about real numbers are limited to those that can be defined in terms of addition, multiplication and ordering. We can implement real numbers in different ways — as Dedekind cuts or Cauchy sequences. However, these different implementations provide identical behavior as viewed through the interface — the different implementations are isomorphic. The axioms of real analysis specify the reals up to isomorphism.

A third hallmark of mathematical maturity is the ability to recognize the existence, or non-existence, of canonical objects. For example, there is no canonical basis for a vector space — any two choices of basis are isomorphic and hence indistinguishable by vector-space properties. There is also no canonical inner product operation on a vector space and no canonical isomorphism of a vector space with its dual. Mathematically mature people understand the general concept of isomorphism, understand that isomorphic objects are intersubstitutable, and understand that when different choices are isomorphic to each other no canonical choice exists.

A fourth hallmark of mathematical maturity is the ability to recognize equivalent definitions. For example, a group can either be defined as a pair of a set and a binary group operator satisfying certain existential statements, e.g., that an inverse operation exists, or as a four tuple of a set, a binary operator, an inverse operator, and an identity element satisfying certain equations. A mathematically mature person recognizes that these definitions are equivalent and the choice is of no consequence.

Most mathematicians today recognize Zermelo-Fraenkel set theory with the axiom of choice (ZFC) as the foundation of mathematics. Unfortunately, ZFC does not illuminate the working mathematicians notion of a well formed statement. It is common in set theory to define the natural numbers by identifying zero with the empty set and identifiing $n + 1$ with the set containing the single element n. We then have, for example, $4 \in 5$. But any working mathematician would agree that $4 \in 5$ is not a well formed statement of arithmetic — it violates the abstraction barrier, or abstract interface, normally associated with the natural numbers.

The second half of this book develops a variety of case studies in types, isomorphism, and type equivalence. The case studies have been selected for their relevance to the field of machine learning. Linear algebra is central to the mathematics of machine learning. Using the general notion of isomorphism developed in the first half of the book we formally prove that a vector space has no canonical basis (no canonical coordinate system), no canonical inner product, and no canonical isomorphism with its dual. We also consider the question of which well-known vector-space algorithms of machine learning require

an ambient inner product and which do not. Newton's method, least squares regression, covariance matrices, the central limit theorem, and canonical correlation analysis (CCA) do not assume any ambient inner product. Principle component analysis (PCA), gradient descent, and L_2 regularization all assume an ambient inner product. There is a also a presentation of Hilbert spaces and Banach spaces. The book ends with a consideration of invariance to nonlinear coordinate transformations including a presentation of differentiable manifolds, Riemannian manifolds, information geometry, natural gradient descent, and Jeffery's prior.

This book gives very little discussion of related work or references to recent type-theoretic literature. Most literature related to the first part of the book involves type systems for computer programming languages. A discussion of programming languages would be a distraction from the central theme of formalizing aspects of mathematical maturity. The case studies in the second part of the book present quite standard elementary material and references can be found easily found elsewhere if desired.

Although this book has only limited references to related work, some basic historical remarks are in order. Types were central to Whitehead and Russell's Principia Mathematica published in 1910. This type system was motivated by the need to avoid set-theoretic paradoxes (Russell's paradox) and did not establish a notion of isomorphsm or a principle of substition of isomorphics. The type-theoretic approach to avoiding paradoxes was supplanted by Zermello-Fraenkel set theory with the axiom of Choice (ZFC) which appeared in its final form in the early 1920's. While ZFC is generally accepted today as capturing the fundamental principles of mathematical proofs, as mentioned above it is untyped and does not provide abstraction barriers preventing access to implementation details.

The intuitively clear fact that a vector space does not have a canonical basis, or a canonical inner product, has come to be formally explained with the concepts of category theory. The desire to formalize the non-existence of a canonical isomorphism between a vector space and its dual was given in the opening paragraph of the original 1945 Eilenberg and MacLane paper introducing category theory. However, the category-theoretic formulation of such statements is not derived from type-theoretic abstraction barriers imposed by the type-theoretic definition of the objects involved. Category theory plays no role in this book.

While axiomatic set theory largely supplanted type theory as a foundation of mathematics for most working mathematicians, type theory has continued to evolve as an alternative foundation. This evolution has largely focused on constructive mathematics starting from a 1934 observation by Curry that implication formulas $P \Rightarrow Q$ (that statement P implies statement Q) can be placed in correspondence with function types $\sigma \to \tau$ (the type of a function mapping objects of type σ to objects of type τ). This has become known as the Curry-Howard isomorphism or the formulas-as-types paradigm. The Curry-Howard isomorphism for propositional logic was extended to a full constructive foundation for mathematics by Coquand and Huet in 1988. The resulting system is called the calculus of construction (COC) and is the foundation of the Coq

interactive theorem prover which is in wide use as of this writing.

The constructive mathematics underling the axiomatic system COC, and the software system Coq, requires that in proving the existence of an object one must give an explicit method of constructing or exhibiting it. While most mathematicians agree that constructive proofs provide more information than non-constructive proofs, non-constructive proofs are still generally accepted as (also) valid and most working mathematicians today see no reason to disallow them. In contrast to the formulas-as-types paradigm of constructive mathematics, Boolean type theory adopts a classical formulas-as-Booleans paradigm. The formulas-as-Booleans paradigm imposes a Boolean abstraction barrier on formulas — a formula denotes a truth value and hence provides only a single bit of information. This Boolean abstraction barrier on formulas appears to be essential to the principle of substitution of isomorphics. Constructive mathematics and the formulas-as-types paradigm play no role in this book.

Types are clearly important in computer programming and type theory has evolved that context. In 1983 John Reynolds published his abstraction theorem which is related to the principle of substitution of isomorphics developed here. But Reynolds' abstraction theorem is based on the notion of a logical relation rather than the working mathematician's notion of isomorphism. Logical relations arise in programming because programming languages do not support equality at function types or universal or existential quantification at infinite types. Logical relations are in one way more general than isomorphisms — they handle the case where an interface disallows equality and/or quantification. But in another respect logical relations are less general — they do not properly handle the form of structure types (dependent sum types) found in working mathematics.

This book is not the final word in type-theoretical mathematical foundations. A more complete treatment would handle homomorphisms and Reynolds' notion of logical relation as well as isomorphism. This book is intended to cover only the most fundamental aspects of general mathematical maturity.

Part I

Foundations

Part I of this book gives a type-theoretic foundation for mathematics. At a high level the foundation is simple — there are only types, truth values, functions and structures. The most fundamental type is **bool**. Expressions of type **bool** denote one of the values "true" or "false". For example the expression as $2 < 5$ is true while the expression $2 > 5$ is false. A function is a mapping from an input type to an output type. If σ and τ are types then $\sigma \to \tau$ is the type whose instances are functions from instances of σ to instances of τ. The type $\mathbf{bool} \to \mathbf{bool}$ is the type of functions mapping a truth value to a truth value. A structure is an object that assigns values to symbolic "slots". For example, a pair of u and v will be written as $\langle \mathbf{first} \leftarrow u; \mathbf{second} \leftarrow v \rangle$. If p is the pair $\langle \mathbf{first} \leftarrow u; \mathbf{second} \leftarrow v \rangle$ then $p.\mathbf{first}$ equals u and $p.\mathbf{second}$ equals v. If τ and σ are types then we write $\overline{\mathbf{first} : \tau;\ \mathbf{second} : \sigma}$ for the type whose instances are pairs whose first component is a τ and whose second component is a σ. Finally we have the type **type** where the instances of the type **type** are themselves types. This allows us to define structures with slots containing types. For example, the structure type $\overline{\alpha : \mathbf{type};\ f : \alpha \to \alpha}$ is the type of structures M which assign values to the symbols α and f and where $M.\alpha$ is a type and $M.f$ is a function of type $M.\alpha \to M.\alpha$.

The first part of this book develops the notions of type, truth value, function, structure, structure isomorphism and the principle of substitution of isomorphics.

Chapter 1

Boolean Logic

As a foundation for mathematics, Boolean type theory is defined by inference rules. These inference rules formally define a general notion of a mathematical proof. As in high school geometry, a proof in mathemamtics generally is a sequence of lines where each line is derivable from previous lines by an application of an inference rule. The inference rules defining Boolean type theory are written in red in figures. For example figure 1.1 gives rules for introducing types and variable declarations. Figure 1.2 gives the inference rules of Boolean logic.

Boolean type theory involves types, truth values, functions and structures. This chapter introduces types and truth values.

1.1 Types, Sequents and Inference Rules

An expression of the form $e : \tau$ states that the expression e has type τ. For example, using $\mathbb{Z}$ to represent the type of integers, we have that $x{:}\mathbb{Z}$ states that the variable x has type integer. If σ and τ are types then $\sigma \to \tau$ is the type of functions from σ to τ. For example $f{:}\mathbb{Z} \to \mathbb{Z}$ says that f is a function mapping integers to integers. A sequent is an expression of the form $\Sigma \vdash \Phi$ where the symbol $\vdash$ can be read as "implies". For example we might have the following where $\mathbb{Z}$ is the type of integers.

$$x{:}\mathbb{Z};\ f{:}\mathbb{Z} \to \mathbb{Z} \vdash f(x){:}\mathbb{Z}$$

This says that **if** x is an integer, and f is a function from integers to integers, **then** $f(x)$ is an integer. In general $\Sigma \vdash \Theta$ can be read as "if Σ then Θ".

Boolean type theory consists of inference rules. These inference rules define both the notion of well-formedness and the notion of proof. For example, the following is a special case of a rule for constructing well formed terms.

$$\frac{\Sigma \vdash e{:}\mathbb{Z} \qquad \Sigma \vdash f{:}\mathbb{Z} \to \mathbb{Z}}{\Sigma \vdash f(e){:}\mathbb{Z}}$$

In general an inference rule consists of a series of sequents above the line called antecedents and a series of sequents below the line called conclusions. A rule states that if we can derive each antecedent sequent then we can also derive each conclusion sequent. A rule with two conclusions is equivalent to two rules — one rule for each conclusion where both rules have the same set of antecedents.

The above rule states that if, in the context Σ, one can derive that e is an integer, and also that f is a function from integers to integers, then one can derive that $f(e)$ is an integer.

1.2 Type Variables

The following sequent avoids the use of the pre-defined type **INT** by declaring a type variable.

$$\alpha{:}\mathbf{type};\ x{:}\alpha;\ f{:}\alpha \to \alpha \vdash f(x){:}\alpha$$

This sequent asserts that if α is a type, x is an α, and f is a function from α to α, then $f(x)$ is also an α. In this sequent each of α, x and f are variables. Each variables is declared to be of a certain type. The variable α is called a type variable and is declared to range over types. The variable x ranges over instances of the type α and the variable f ranges over functions from α to α.

Here the symbol **type** is itself being used as a type — it is the type of the variable α. We are tempted to write $\mathbf{type} : \mathbf{type}$. Unfortunately this assertion leads to a form of Russell's paradox — the inference system becomes inconsistent. To avoid paradoxes we write $\mathbf{type}_0 : \mathbf{type}_1$ and more generally $\mathbf{type}_i{:}\mathbf{type}_j$ for $j > i$. All occurrences of the symbol **type** must be subscripted and we must rewrite the above sequent as, for example, the following.

$$\alpha{:}\mathbf{type}_0;\ x{:}\alpha;\ f{:}\alpha \to \alpha \vdash f(x){:}\alpha$$

The type $\mathbf{type}_0$ is an instance of the type $\mathbf{type}_1$; $\mathbf{type}_1$ is an instance of $\mathbf{type}_2$; and so on. We take $\mathbf{type}_0$ to contain only finite types. An inference rule expressing the finiteness of types in $\mathbf{type}_0$ is given in section 3.4.

1.3 Rules for Declaring Variables

Figure 1.1 gives rules for declaring variables. In the base case rule we use ε to denote the empty context. A derivation is a sequence of numbered lines where each line is derived from previous lines using an inference rule. For example, we have the following.

1	ε	$\vdash$	$\mathbf{type}_1{:}\mathbf{type}_2$	Base Case Rule
2	$\varepsilon;\ \alpha{:}\mathbf{type}_1$	$\vdash$	$\alpha{:}\mathbf{type}_1$	type var. declaration from 1
3	$\varepsilon;\ \alpha{:}\mathbf{type}_1;\ x{:}\alpha$	$\vdash$	$x{:}\alpha$	term var. declaration from 2
4	$\varepsilon;\ \alpha{:}\mathbf{type}_1;\ x{:}\alpha$	$\vdash$	$\alpha{:}\mathbf{type}_1$	judgement promotion from 3,2
5	$\varepsilon;\ \alpha{:}\mathbf{type}_1;\ x{:}\alpha;\ y{:}\alpha$	$\vdash$	$y{:}\alpha$	term var. declaration from 4

Base Case Rule:

$$\frac{j > i}{\varepsilon \vdash \mathbf{type}_i : \mathbf{type}_j}$$

Variable Declaration:

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad x \text{ is a variable not in } \Sigma}{\Sigma\ ;\ x : \tau \vdash x : \tau}$$

Judgement Promotion:

$$\frac{\Sigma \vdash \Phi \quad \Sigma\ ;\ \Psi \vdash \Psi}{\Sigma\ ;\ \Psi \vdash \Phi}$$

Type Promotion:

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad j > i}{\Sigma \vdash \tau : \mathbf{type}_j}$$

Figure 1.1: **Declaring Variables**.

1.4 Rules for Boolean Logic

The type **bool** is the type of well formed statements. Expressions of type **bool** are called formulas. Formulas have truth values — if we assign a meaning to all the variables declared in Σ consistent with their declared types then every formula constructed from those variables has a well defined truth value. Inference rules for Boolean logic are given in figure 1.2. The following derivation is an example of the application of these rules. The empty context ε is not shown.

1		$\vdash$	$\mathbf{bool} : \mathbf{type}_1$	boolean type formation
2	$P : \mathbf{bool}$	$\vdash$	$P : \mathbf{bool}$	term var. declaration from 1
3	$P : \mathbf{bool}$	$\vdash$	$\neg P : \mathbf{bool}$	negation formation from 2
4	$P : \mathbf{bool};\ P$	$\vdash$	P	assumption formation from 2
5	$P : \mathbf{bool};\ P$	$\vdash$	$P \vee \neg P$	disj. deriv. from 4,3
6	$P : \mathbf{bool};\ \neg P$	$\vdash$	$\neg P$	assumption formation from 3
7	$P : \mathbf{bool};\ \neg P$	$\vdash$	$\neg P \vee P$	disj deriv. from 6,2
8	$P : \mathbf{bool};\ \neg P$	$\vdash$	$P \vee \neg P$	disj symmetry from 7
9	$P : \mathbf{bool}$	$\vdash$	$P \vee \neg P$	case analysis from 5,8

In some cases a derivation will start with one or more unjustified lines as in the following. Judgement promotion steps are not shown.

Boolean Type Formation:

$$\frac{}{\varepsilon \vdash \mathbf{bool} : \mathbf{type}_0}$$

Assumption Formation:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool}}{\Sigma \,;\, \Phi \vdash \Phi}$$

Disjunction Formation:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool} \quad \Sigma \vdash \Psi : \mathbf{bool}}{\Sigma \vdash (\Phi \vee \Psi) : \mathbf{bool}}$$

Disjunction Derivation:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool} \quad \Sigma \vdash \Psi : \mathbf{bool} \quad \Sigma \vdash \Phi}{\Sigma \vdash \Phi \vee \Psi}$$

Disjunction Symmetry:

$$\frac{\Sigma \vdash \Psi \vee \Phi}{\Sigma \vdash \Phi \vee \Psi}$$

Conjunction Derivation:

$$\frac{\Sigma \vdash \neg\Phi \quad \Sigma \vdash \neg\Psi}{\Sigma \vdash \neg(\Phi \vee \Psi)}$$

Negation Formation:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool}}{\Sigma \vdash \neg\Phi : \mathbf{bool}}$$

Double Negation:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool} \quad \Sigma \vdash \Phi}{\Sigma \vdash \neg\neg\Phi}$$

Case Analysis:

$$\frac{\Sigma \,;\, \Phi \vdash \Psi \quad \Sigma \,;\, \neg\Phi \vdash \Psi}{\Sigma \vdash \Psi}$$

Cut:

$$\frac{\Sigma \vdash \neg\Phi \quad \Sigma \vdash \Phi \vee \Psi}{\Sigma \vdash \Psi}$$

Figure 1.2: **Inference Rules for Boolean Logic**. Here we take $\vee$ (or) and $\neg$ (not) to be primitive and treat $\Phi \wedge \Psi$ (and) as an abbreviation for $\neg(\neg\Phi \vee \neg\Psi)$; $\Phi \Rightarrow \Psi$ (implies) as an abbreviation for $\neg\Phi \vee \Psi$ and $\Phi \Leftrightarrow \Psi$ (if and only if) as an abbreviation for $(\Phi \Rightarrow \Psi) \wedge (\Psi \Rightarrow \Phi)$.

1	Σ	$\vdash$	$\Phi : \mathbf{bool}$	hypothesis
2	Σ	$\vdash$	$\Psi : \mathbf{bool}$	hypothesis
3	$\Sigma;\ \Phi$	$\vdash$	Ψ	hypothesis
4	$\Sigma;\ \Phi$	$\vdash$	$\neg\Phi : \mathbf{bool}$	negation formation from 1
5	$\Sigma;\ \Phi$	$\vdash$	$\Psi \vee \neg\Phi$	disjunction derivation from 3,4
6	$\Sigma;\ \Phi$	$\vdash$	$\neg\Phi \vee \Psi$	disjunction symmetry from 5
7	Σ	$\vdash$	$\neg\Phi : \mathbf{bool}$	negation formation from 1
8	$\Sigma;\ \neg\Phi$	$\vdash$	$\neg\Phi$	assumption formation from 7
9	$\Sigma;\ \neg\Phi$	$\vdash$	$\neg\Phi \vee \Psi$	disjunction derivation from 8,2
10	Σ	$\vdash$	$\neg\Phi \vee \Psi$	case analysis from 6,9
11	Σ	$\vdash$	$\Phi \Rightarrow \Psi$	same as 10

This derivation justifies the following derived rule.

Implication Derivation:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool} \quad \Sigma \vdash \Psi : \mathbf{bool} \quad \Sigma;\ \Phi \vdash \Psi}{\Sigma \vdash \Phi \Rightarrow \Psi}$$

Problem 1.1: Derive the following rule from the fact that implication is an abbreviaiton in terms of disjunction and negation, the double negation rule and the cut rule.

Modus Ponens:

$$\frac{\Sigma \vdash \Phi \Rightarrow \Psi \quad \Sigma \vdash \Phi}{\Sigma \vdash \Psi}$$

Problem 1.2: Use the implication derivation rule and the cut rule to derive the following contrapositive rule.

Contrapositive:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool} \quad \Sigma \vdash \Psi : \mathbf{bool} \quad \Sigma;\ \Phi \vdash \Psi}{\Sigma;\ \neg\Psi \vdash \neg\Phi}$$

Problem 1.3: Use the contrapositive rule and the case analysis rule to derive the following contradiction rule.

Contradiction:

$$\frac{\Sigma \vdash \Phi : \mathbf{bool} \quad \Sigma \vdash \Psi : \mathbf{bool} \quad \Sigma;\ \Phi \vdash \Psi \quad \Sigma;\ \Phi \vdash \neg\Psi}{\Sigma \vdash \neg\Phi}$$

Chapter 2

Functions and Quantification

Boolean type theory involves types, truth values, functions and structures. The previous chapter introduced types and truth values (Boolean expressions). This chapter introduces functions, absolute equality and universal and existential quantification.

2.1 Functions

Figure 2.1 gives rules for forming function types and application terms. In this section we consider only the two rules of function type formation and function variable. The following derivation is an example of the application of these two rules. This derivation omits some steps — see the comments in the next section.

1	$\alpha{:}\mathbf{type}_1$	$\vdash$	$\alpha{:}\mathbf{type}_1$	type var. declaration
2	$\alpha{:}\mathbf{type}_1$	$\vdash$	$(\alpha \to \alpha){:}\mathbf{type}_1$	function type form. from 1
3	$\alpha{:}\mathbf{type}_1;\ f{:}\alpha \to \alpha$	$\vdash$	$f{:}\alpha \to \alpha$	var. declaration from 2
4	$\alpha{:}\mathbf{type}_1;\ f{:}\alpha \to \alpha;\ x{:}\alpha$	$\vdash$	$x{:}\alpha$	var. declaration from 1
5	$\alpha{:}\mathbf{type}_1;\ f{:}\alpha \to \alpha;\ x{:}\alpha$	$\vdash$	$f(x){:}\alpha$	function variable from 4,3

2.2 Abbreviated Derivations

It is not practical to write every step in derivations when we require that every step be justified from previous steps by a single rule of inference. In practice we give derivations as a sequence of lines where each line is justified by listing previous lines (or previously proved theorems) from which the new line can be derived in a straightforward but perhaps tedious way. The derivation in the preceding section can be given as follows where we have deleted references to particular rules of Boolean type theory.

1	$\alpha{:}\mathbf{type}_1$	$\vdash$	$\alpha{:}\mathbf{type}_1$	
2	$\alpha{:}\mathbf{type}_1$	$\vdash$	$(\alpha \to \alpha){:}\mathbf{type}_1$	1
3	$\alpha{:}\mathbf{type}_1;\ f{:}\alpha \to \alpha$	$\vdash$	$f{:}\alpha \to \alpha$	2
4	$\alpha{:}\mathbf{type}_1;\ f{:}\alpha \to \alpha;\ x{:}\alpha$	$\vdash$	$x{:}\alpha$	1
5	$\alpha{:}\mathbf{type}_1;\ f{:}\alpha \to \alpha;\ x{:}\alpha$	$\vdash$	$f(x){:}\alpha$	4,3

Given that some experience has been established with the basic rules in the above derivation future derivations can be more abbreviated. For example we might write the following derivation.

1	$\Sigma; x{:}\sigma; f{:}\sigma \to \sigma;\ g{:}\sigma \to \tau$	$\vdash$	$f(x){:}\sigma$	
2	$\Sigma; x{:}\sigma; f{:}\sigma \to \sigma;\ g{:}\sigma \to \tau$	$\vdash$	$f(f(x)){:}\sigma$	1
3	$\Sigma; x{:}\sigma; f{:}\sigma \to \sigma;\ g{:}\sigma \to \tau$	$\vdash$	$f(f(f(x))){:}\sigma$	2
4	$\Sigma; x{:}\sigma; f{:}\sigma \to \sigma;\ g{:}\sigma \to \tau$	$\vdash$	$g(f(f(f(x)))){:}\tau$	3

The judgement as to whether a given step is straightforward is subjective. When giving a proof in a published paper the author must anticipate the reader's notion of straightforwardness. When giving derivations on problem sets or exams one must try to predict the grader's notion of straightforwardness. A problem set or exam that asks a student to give a proof (a derivation) should, in part at least, test the students ability to give the proof at an appropriate level of detail.

2.3 Variable Substitutions

The second two rules in figure 2.1 involve lambda expressions. A lambda expression is a way of naming a particular function. For example the expression $(\lambda\ x : \mathbb{R}\ x + x)$, where $\mathbb{R}$ is the type of real numbers, denotes the function f from real numbers to real numbers satisfying $f(x) = x + x$. More generally $(\lambda\ x : \tau\ e[x])$ denotes the function f such that for any x of type τ we have $f(x) = e[x]$.

To formally define the inference rules for lambda expressions we need to formally explain the notation $e[x]$ and $e[s]$. This involves formally describing the notions of free and bound variables and variable substitution. Although we have not yet introduced universal and existential quantifiers, the notions of free and bound variable are perhaps most naturally discussed in terms of these quantifiers. The formula $\forall x : \tau\ \ \Phi[x]$ expresses the statement that for all x of type τ we have that $\Phi[x]$ holds where $\Phi[x]$ is a Boolean expression (a formula) involving the variable x. The formula $\exists x : \tau\ \ \Phi[x]$ expresses the statement that there exists an x of type τ such that $\Phi[x]$ holds.

The meaning of an expression depends on the meaning of its free variables. For example, let x and y be variables ranging over natural numbers. The formula $x = y$ is true for some assignments of values to x and y and false for other assignments. But, given an asignment of values to both x and y, the formula $x = y$ has a well defined truth value. Now consider a formula of the form $\exists x : \tau\ \ \Phi[x, y]$. In this formula y is a free variable — the truth of the formula

can depend on the value of y. For example consider $\exists x : \mathbb{N} \;\; x < y$ where $\mathbb{N}$ is type of natural numbers (non-negative integers). This formula is true when y is greater than zero but false when y is zero. However, x, unlike y, is not a free variable of the formula $\exists x : \tau \;\; \Phi[x, y]$ — to determine the value of the formula we need only specify the value of y. The occurance of x in this formula is said to be *bound*. More precisely, an occurance of a variable x in an expression e is said to be bound if that occurance occurs inside a quantification over x.

If e and u are expressions and x is a variable we write $e[x \leftarrow u]$ to denote the expression which results from replacing all free occurrences of x in e with u with renaming of bound variables in e to avoid the capture of free variables in u. To see the need for the "renaming" clause consider the following.

$$(\exists x : \mathbb{Z}\ y = 2 * x)[y \leftarrow 3 * x]$$

This is the same as the following.

$$(\exists z : \mathbb{Z}\ y = 2 * z)[y \leftarrow 3 * x]$$

This is the renaming step. We have renamed the bound variable x to z. This does not change the meaning of the existential formula. This renaming avoids "capturing" x when the substitution is done. The results of the substitution is then the following.

$$\exists z : \mathbb{Z}\ 3 * x = 2 * z$$

Without the renaming we would have gotten $\exists x : \mathbb{Z}\ 3 * x = 2 * x$ which is not what we want.

> **Problem** 2.1: Give the result of the subsitution $(\forall x : \tau\ x = y)[y \leftarrow x]$.

We will adopt the convention that when an expression e is first written as $e[x]$ and then later written as $e[u]$ the latter occurrence is just an abbreviation for the substitution $e[x \leftarrow u]$.

Lambda expressions also bind variables. Consider $(\lambda\ x : \mathbb{N} \;\; x + y)$. This is the function f_y such that $f_y(x) = x + y$. The meaning of this lambda expression depends on the value of y (which is a free variable of the expression). But we do not need to specify any particular value for x in determining the meaning of the lambda expression $(\lambda\ x : \mathbb{N} \;\; x + y)$ — the function f_y denoted by this expression is determined by the value of y. In general a lambda expression $(\lambda\ x : \tau\ e[x])$ quantifies over the variable x so that all occurrances of x in $e[x]$ are bound by the lambda expression.

> **Problem** 2.2: Give the result of the subsitution $(\lambda x : \mathbb{N}\ x + y)[y \leftarrow x]$.

2.4 Functions of More than One Argument

Functions of more than one argument can be modeled by Currying. More specifically, consider a function f of two arguments and consider the term $f(x, y)$. Rather than use functions of two arguments we can define a function f' such that $f'(x)$ is a function which maps y to $f(x, y)$. We then have that $f(x, y)$ can be written as $f'(x)(y)$. In general a function f taking two arguments where the first has type τ_1 and the second has type τ_2, and where f produces a value of type σ, can be represented by a function $f' : \ \tau_1 \to (\tau_2 \to \sigma)$. Functions of three or more arguments can be handles similarly. We will sometimes write $\tau_1 \times \tau_2 \to \sigma$ as an alternate notation for $\tau_1 \to (\tau_2 \to \sigma)$ and $f(x, y)$ as an alternate notation for $f(x)(y)$. Using these notations we have the following.

$$\alpha{:}\mathbf{type}_1;\ \beta{:}\mathbf{type}_1;\ x{:}\alpha;\ f{:}\alpha \to \alpha;\ g{:}\alpha \times \alpha \to \beta \quad \vdash \quad g(x, f(x)){:}\beta$$

We will also use an abbreviated notion for lambda expressions of more than one argument. The expression $(\lambda\ (x : \sigma,\ \ y : \tau)\ e[x, y])$ will be used as an abbreviation for $(\lambda\, x{:}\sigma\ (\lambda\, y{:}\tau\ e[x, y]))$.

2.5 Absolute Equality

The inference rule of beta reduction in figure 2.1 derives an equation. Inference rules for equality are given in figure 2.2. Later chapters will develop structure types and the notion of isomorphism. For example, we will be able to write the type **Graph** and there will be an associated notion of isomorphism for graphs. We will write $G_1 =_{\mathbf{Graph}} G_2$ to indicate that the graphs G_1 and G_2 are isomorphic as graphs. In general $x =_\tau y$ will mean that x and y are isomorphic as instances of τ. Equations of the form $x =_\tau y$ will be called type-relative equalities. Type-relative equality is developed in chapter 3

The equalities in figures 2.1 and 2.2 are called absolute equalities in contrast to type-relative equalities. As explained in chapter 3, absolute equalities violate abstraction barriers. For this reason absolute equalities are not Boolean expressions — in Boolean type theory it is essential that Boolean expressions not violate abstraction barriers. However, non-Boolean judgements, such as a type judgement $e{:}\mathbb{Z}$, or an absolute equation $x = 5$, play an important role in Boolean type theory in spite of their violation of abstraction barriers.

2.6 Rules for Quantification

Figure 2.3 gives the fundamental rules for the quantifer $\forall$. We treat an existential formula $\exists x{:}\tau\ \Phi[x]$ as an abbreviation for $\neg\forall x{:}\tau\ \neg\Phi[x]$. We will also write $\forall x{:}\tau, y{:}\sigma\ \Phi[x, y]$ as an abbreviation for $\forall x{:}\tau\ \forall y{:}\sigma\ \Phi[x, y]$.

It turns out that it is technically convenient to express universally quantified formulas in terms of predicates. This simplifies the inference rules. Rather than

Function Type Formation:

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma \vdash \sigma : \mathbf{type}_i}{\Sigma \vdash (\tau \to \sigma) : \mathbf{type}_i}$$

Function Variable:

$$\frac{\Sigma \vdash f : (\tau \to \sigma) \quad \Sigma \vdash e : \tau}{\Sigma \vdash f(e) : \sigma}$$

Function Instance Formation:

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma; x : \sigma \vdash e[x] : \tau}{\Sigma \vdash (\lambda x : \sigma\ e[x]) : (\sigma \to \tau)}$$

Beta Reduction:

$$\frac{\Sigma \vdash (\lambda x : \sigma\ e[x]) : \sigma \to \tau \quad \Sigma \vdash s : \sigma}{\Sigma \vdash (\lambda x : \sigma\ e[x])(s) = e[s]}$$

Figure 2.1: **Functions**. Type formation rules, instance formation rules, and variable rules are given for many types. Although the variable rule allows f to be any expression, it is particularly useful when f is a variable of type $\sigma \to \tau$. The beta reduction rule derives an absolute equality. The inference rules for absolute equality are given in figure 2.2.

Reflexivity of Equality:

$$\frac{\Sigma \vdash e : \tau}{\Sigma \vdash e = e}$$

Symmetry of Equality:

$$\frac{\Sigma \vdash s = w}{\Sigma \vdash w = s}$$

Transitivity of Equality:

$$\frac{\Sigma \vdash s = w \quad \Sigma \vdash w = t}{\Sigma \vdash s = t}$$

Absolute Function Extensionality:

$$\frac{\Sigma \vdash f : \sigma \to \tau \quad \Sigma \vdash g : \sigma \to \tau \quad \Sigma; x : \sigma \vdash f(x) = g(x)}{\Sigma \vdash f = g}$$

Absolute Substitution:

$$\frac{\Sigma \vdash s = w \quad \Sigma \vdash \Theta[s]}{\Sigma \vdash \Theta[w]}$$

Figure 2.2: **Absolute Equality.** Absolute equality implies absolute substituability. See chapter 3 for a discussion of the relationship between absolute equality and type-relative equality.

Forall Formula Formation:

$$\frac{\Sigma\,;\vdash P{:}\tau\to\mathbf{bool}}{\Sigma\vdash(\forall\tau\ P){:}\mathbf{bool}}$$

Forall Utilization:

$$\frac{\Sigma\vdash(\forall\tau\ P)\quad \Sigma\vdash e{:}\tau}{\Sigma\vdash P(e)}$$

Forall Derivation:

$$\frac{\Sigma\,;\vdash P{:}\tau\to\mathbf{bool}\quad \Sigma;\ x{:}\tau\vdash P(x)}{\Sigma\vdash(\forall\tau\ P)}$$

Axiom of Choice:

$$\frac{\Sigma\vdash\forall x{:}\tau\ \exists y{:}\sigma\ \Phi[x,\ y]\quad x\text{ does not occur free in }\sigma}{\Sigma\vdash\exists f{:}\tau\to\sigma\ \forall x{:}\tau\ \Phi[x,f(x)]}$$

Figure 2.3: **Rules for Quantification**. We take $\forall x:\tau\Phi[x]$ to be an abbreviation for $(\forall\tau(\lambda x:\tau\ \Phi[x]))$. We use universal quantification as basic and treat the formula $\exists x:\tau\ \Phi[x]$ as an abbreviation for $\neg\forall x:\tau\ \neg\Phi[x]$.

write $\forall x:\tau\ \Phi[x]$ we write $(\forall\tau(\lambda x:\tau\Phi[x]))$. This latter expression has the form $(\forall\tau P)$ where P is a predicate on τ, i.e., P has type $\tau\to\mathbf{bool}$. The inference rules are written in using predicates. These "simplified" rules have the property that variables are bound by the lambda quantifiers inside predicates rather than the universal and existential quantifiers of Boolean formulas.

Problem 2.3: Derive the following inference rules from those given in figure 2.3. In the existential utilization the second and third antecedent together imply that x does not occur in Ψ.

Existential Derivation:

$$\frac{\Sigma\ ;\ x{:}\tau\vdash\Phi[x]{:}\mathbf{bool}\quad \Sigma\vdash e{:}\tau\quad \Sigma\vdash\Phi[e]}{\Sigma\vdash(\exists x{:}\tau\ \Phi[x])}$$

Existential Utilization:

$$\frac{\Sigma\vdash\exists x{:}\tau\ \Phi[x]\quad \Sigma\ ;\ x{:}\tau;\ \Phi[x]\vdash\Psi\quad \Sigma\vdash\Psi{:}\mathbf{bool}}{\Sigma\vdash\Psi}$$

Chapter 3

Type-Relative Equality

One of our main objective is to enforce the abstraction barriers imposed by types — well formed statements should not be able to distinguish isomorphic implementations. It turns out that preserving abstraction barriers requires careful design of the inference rules. Consider the following problematic (and ill-formed) predicate on graphs.

$$(\lambda G{:}\mathbf{Graph}\ \ \exists n{:}G.\mathbf{node}\ \ n = 5) : \mathbf{Graph} \to \mathbf{bool} \quad \text{is ill-formed}$$

If we can ask whether a general graph has a node equal to the number 5 then we can distinguish isomorphic graphs. The equation $n = 5$ violates the abstraction barrier. The following predicate is similarly problematic (and ill-formed).

$$(\lambda G{:}\mathbf{Graph}\ \ \exists n{:}G.\mathbf{node}\ \ n{:}\mathbb{Z}) : \mathbf{Graph} \to \mathbf{bool} \quad \text{is ill-formed}$$

These predicates are ill-formed because absolute equalities $u = w$ and type judgements $e : \sigma$ are not Boolean expressions. In Boolean type theory it is essential that Boolean expressions not violate abstraction barriers.

In addition to absolute equalities, which violate abstraction barriers and are not Boolean expressions, it is important to also have equalities that respect abstraction barriers. This can be done by making equality type-relative. We write $s =_\tau w$ to mean that s is equal to w when s and w are considered as elements of the type τ. If we have declared a type variable $\alpha : \mathbf{type}_i$ then we can write equalities $x =_\alpha y$ between instances x and y of type α. This allows us to write equalities between nodes in a given graph. Equalities between nodes of a given graph do not violate the graph abstraction barrier. Also, the equation $G =_{\mathbf{Graph}} G'$ states that G and G' are isomorphic as graphs. This equation also respects the graph abstraction barrier. Type-relative equalities are well formed Boolean expressions.

This chapter also introduces subtypes. Subtypes play an important role in Boolean type theory. For example, a standard way of constructing the real numbers is to implement the type of real numbers as a subtype of $\mathcal{Q} \to \mathbf{bool}$ (Dedekind cuts) where $\mathcal{Q}$ is the type of rationals. It is convenient to introduce

Subtype Type Formation:

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma \vdash P : \tau \to \mathbf{bool}}{\Sigma \vdash \mathbf{TheSubtype}(\tau, P) : \mathbf{type}_i}$$

Subtype Variable:

$$\frac{\Sigma \vdash e : \mathbf{TheSubtype}(\tau, P)}{\Sigma \vdash e : \tau \quad \Sigma \vdash P(e)}$$

Subtype Instance Formation:

$$\frac{\Sigma \vdash e : \tau \quad \Sigma \vdash P : \tau \to \mathbf{bool} \quad \Sigma \vdash P(e)}{\Sigma \vdash e : \mathbf{TheSubtype}(\tau, P)}$$

Figure 3.1: **Rules for Subtypes.**

subtypes before giving the rules for type-relative equality — using subtypes we can define the type $\mathbf{Bijection}[\sigma, \tau]$ which is used in the inference rules for type-relative equality at the type $\mathbf{type}_i$.

This chapter also introduces definite description of the form $\mathbf{The}(x{:}\tau\ \Phi[x])$ which the "the" x of type τ satisfying the condition $\Phi[x]$. Definite descriptions are straightforward and convenient.

3.1 Subtypes

If τ is a type and P is a predicate on τ, i.e., a function of type $\tau \to \mathbf{bool}$, then we write $\mathbf{TheSubtype}(\tau, P)$ for the type whose instances are those instances x of τ such that $P(x)$ is true and where the equivalence relation on $\mathbf{TheSubtype}(\tau, P)$ is just the restriction of the equivalence relation on τ to the instances of the subtype. Rules for subtypes are given in figure 3.1.

The following abbreviation is often convenient.

$$\mathbf{TheSubtype}\,(x{:}\tau \mid \Phi[x]) \equiv \mathbf{TheSubtype}\,(\tau,\ (\lambda x{:}\tau \mid \Phi[x]{:}\mathbf{bool}))$$

As an example of a subtype we introduce the following notion of a bijection.

$$\mathbf{Bijection}[\alpha, \beta] \quad \equiv \quad \mathbf{TheSubtype}\,(f{:}\alpha \to \beta \mid \forall\, y{:}\beta\ \exists!\ x{:}\alpha\ y =_\beta f(x))$$

$$\exists! x{:}\tau\ \Phi[x] \quad \equiv \quad (\exists x{:}\tau\ \Phi[x]) \wedge (\forall x{:}\tau, y{:}\tau\ \ (\Phi[x] \wedge \Phi[y] \Rightarrow x =_\tau y))$$

We can then write a variable declaration such as $f{:}\mathbf{Bijection}[\alpha, \beta]$.

Problem 3.1: An injection from σ to τ is a function $f : \sigma \to \tau$ such that distinct inputs (as defined by $=_\sigma$) yield distinct outputs

Relative Equality is Boolean:

$$\frac{\Sigma \vdash s : \sigma \quad \Sigma \vdash w : \sigma}{\Sigma \vdash (s =_\sigma w) : \mathbf{bool}}$$

Reflexivity:

$$\frac{\Sigma \vdash e : \tau}{\Sigma \vdash e =_\tau e}$$

Symmetry:

$$\frac{\Sigma \vdash e =_\tau w}{\Sigma \vdash w =_\tau e}$$

Transitivity:

$$\frac{\Sigma \vdash e =_\tau w \quad \Sigma \vdash w =_\tau s}{\Sigma \vdash e =_\tau s}$$

Equality at $\mathbf{type}_i$:

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma \vdash \sigma : \mathbf{type}_i \quad \Sigma \vdash \exists\, \mathbf{Bijection}[\tau, \sigma]}{\Sigma \vdash \sigma =_{\mathbf{type}_i} \tau}$$

Equality at Function Types:

$$\frac{\Sigma \vdash f : \sigma \to \tau \quad \Sigma \vdash g : \sigma \to \tau \quad \Sigma \vdash \forall x : \sigma\ f(x) =_\tau g(x)}{\Sigma \vdash f =_{\sigma \to \tau} g}$$

Equality at Subtypes:

$$\frac{\Sigma \vdash s : \mathbf{TheSubtype}(\tau,\ P) \quad \Sigma \vdash w : \mathbf{TheSubtype}(\tau,\ P) \quad \Sigma \vdash s =_\tau w}{\Sigma \vdash s =_{\mathbf{TheSubtype}(\tau,\ P)} w}$$

Substitution:

$$\frac{\Sigma \vdash f : \sigma \to \tau \quad \Sigma \vdash s =_\sigma w}{\Sigma \vdash f(s) =_\tau f(w)}$$

Figure 3.2: **Type-Relative Equality.** Equalities between Booleans can be derived from the reflexivity rule. Negations of equalities are proved using proof by contradiction.

(as defined by $=_\tau$). Assuming that σ and τ are well formed types, give a well-formed expression for the type Injection$[\sigma, \tau]$. Hint: you have to use a subtype.

3.2 Type-Relative Equality

The inference rules for type-relative equality are shown in figure 3.2. The following series of problems give important properties of type-relative equality. It is convenient at this point to introduce the Boolean constants **True** and **False** as follows.

$$\begin{aligned} \mathbf{True} &\equiv \exists P{:}\mathbf{bool}\ P \\ \mathbf{False} &\equiv \forall P{:}\mathbf{bool}\ P \end{aligned}$$

Problem 3.2: Give a derivations of $\vdash$ **True** and $\vdash \neg$**False**.

Problem 3.3: Give a derivations of $\vdash$ **True** $=_{\mathbf{bool}}$ **True** and $\vdash$ **False** $=_{\mathbf{bool}}$ **False**.

Problem 3.4: Give a derivation of $\vdash$ **True** $\neq_{\mathbf{bool}}$ **False**. (Hint: Use proof by contradiction and substitution).

Problem 3.5: Given a derivation of $\vdash \forall P{:}\mathbf{bool}\ (P =_{\mathbf{bool}}$ **True**$) \vee (P =_{\mathbf{bool}}$ **False**).

Problem 3.6: Derive the following inference rule.

Disequation Symmetry:

$$\frac{\Sigma \vdash e \neq_\sigma w}{\Sigma \vdash w \neq_\sigma e}$$

The substitution rule is nontrivial. The formula $G =_{\mathbf{Graph}} G'$ does not imply that G nd G' are the same thing — G and G' are merely isomorphic as graphs. The substitution rules states that no well-formed function (or predicate) defined on an arbitrary graph can distinguish between isomorphic graphs. This is related to the meaning of the function type $\sigma \to \tau$. All instances of this type must respect the equivalence relations $=_\sigma$ and $=_\tau$.

Problem 3.7: Use the substitution rule and lambda expressions to derive the following variant of the substitution rule.

Substitution Variant:

$$\frac{\Sigma \vdash \tau : \mathbf{type} \quad \Sigma;\ x:\sigma \vdash e[x]:\tau \quad \Sigma \vdash s =_\sigma w}{\Sigma \vdash e[s] =_\tau e[w]}$$

Problem 3.8: Derive the substitution rule in figure 3.2 form the above substitution variant rule.

Problem 3.9: Derive the following rule.

Disequation Derivation:

$$\frac{\Sigma \vdash P:\sigma \to \mathbf{bool} \quad \Sigma \vdash P(e) \quad \Sigma \vdash \neg P(w)}{\Sigma \vdash e \neq_\sigma w}$$

The rule for equality at $\mathbf{type}_i$ states that two types are equal (isomorphic as types) if they have the same cardinality (number of elements). The substitution rule then implies that no predicate on types (which must be well formed for an arbitrary input type) cannot distinguish between two types of the same cardinality. For an arbitrary type declaration $\alpha:\mathbf{type}_i$ the type α has no visible structure other than the number of distinct instances.

Problem 3.10: An equivalence relation is defined by reflexivity (everything is equivalent to itself) transitivity and symmetry. A partition of a type σ is a collection of subsets of σ (or predicates on σ) such that every element of σ is contained in exactly one of those subsets.

a. Consider $P : \sigma \times \sigma \to \mathbf{bool}$. Give a well typed Boolean expression involving P which is true if and only if P is an euivalence relation on σ.

b. Consider $Q:(\sigma \to \mathbf{bool}) \to \mathbf{bool}$. Give a well typed Boolean expression involving Q which is true if and only if Q is a partition of σ.

3.3 Definite Descriptions

If we can prove that there exists exactly one object with a given property then it is convenient to have a term denoting that object. For this purpose we introduce definite description terms of the form $\mathbf{The}(\tau,\ P)$ where $P:\tau \to \mathbf{bool}$. The inference rules for definite description terms are given in figure 3.3.

Definite Description Formation:

$$\frac{\Sigma \vdash P : \sigma \to \mathbf{bool} \quad \Sigma \vdash \exists! x : \tau \;\; P(x)}{\Sigma \vdash \mathbf{The}(\tau,\; P) : \tau}$$

Definite Description Utilization:

$$\frac{\Sigma \vdash \mathbf{The}(\tau,\; P) : \tau}{\Sigma \vdash P(\mathbf{The}(\tau,\; P))}$$

Figure 3.3: **Rules for Definite Descriptions**. We will write $\mathbf{The}(x : \tau\ \Phi[x])$ as an abbreviation for $\mathbf{The}(\tau,\ (\lambda x : \tau\ \Phi[x]))$. We will also write $\mathbf{The}(\tau)$ as an abbreviation for $\mathbf{The}(x : \tau\ \mathbf{True})$.

$\mathbf{type}_0$ Induction:

$$\frac{\Sigma \vdash P : \mathbf{type}_0 \to \mathbf{bool} \quad \Sigma \vdash P(\mathbf{EmptyType}) \quad \Sigma;\ \alpha : \mathbf{type}_0;\ P(\alpha) \vdash P(\mathbf{SuccType}[\alpha])}{\Sigma \vdash \forall \alpha : \mathbf{type}_0\ P(\alpha)}$$

Figure 3.4: **Induction Rule for $\mathbf{type}_0$**

Problem 3.11: Give an expression for a function $\mathbf{if} : \mathbf{bool} \times \alpha \times \alpha \to \alpha$ with the property that $\mathbf{if}(\mathbf{\Phi}, \mathbf{x}, \mathbf{y})$ equals x if Φ is true and y otherwise.

3.4 The Induction Rule for $\mathbf{type}_0$

Figure 3.4 gives an inference rule allowing us to infer that types in $\mathbf{type}_0$ are finite. The inference rule is analogous to the induction rule for natural numbers and in fact the natural numbers are constructed from the type $\mathbf{type}_0$ in chapter 7. The rule involves the following notations.

$$\sigma \backslash x \quad \equiv \quad \mathbf{TheSubtype}(y : \sigma\ y \neq_\sigma x)$$

$$\mathbf{EmptyType} \quad \equiv \quad \mathbf{The}(\alpha : \mathbf{type}_0 \mid \neg\exists \alpha)$$

$$\mathbf{SuccType}[\alpha] \quad \equiv \quad \mathbf{The}(\beta : \mathbf{type}_0\ \exists x : \beta\ \exists \mathbf{Bijection}[\alpha, \beta \backslash x])$$

We omit the proof that the above definite descriptions are well formed.

Problem 3.12: Is the predicte $(\lambda G : \mathrm{Graph}\ \exists n : G.\mathbf{node}\ \ n : \sigma)$ a well formed predicate on graphs? Explain your answer.

Chapter 4

Structures

For a well formed context Γ, i.e., a context Γ such that $\Gamma \vdash \mathbf{True}$, we will write $\overline{\Gamma}$ for the type of structures which assign values to the variables declared in Γ in a way that satisfies the declarations and formulas in Γ. For example, we can define the type finite **DiGraph** as follows.

$$\mathbf{DiGraph} \equiv \overline{\mathbf{node}{:}\mathbf{type}_0;\ \mathbf{edge}{:}\ \mathbf{node} \times \mathbf{node} \to \mathbf{bool}}$$

The type of finite undirected graphs can be defined as follows.

$$\begin{array}{rcl} \mathbf{graph} & \equiv & \overline{\mathbf{node}{:}\mathbf{type}_0;\ \mathbf{edge}{:}\ \mathbf{node} \times \mathbf{node} \to \mathbf{bool};\ \Phi} \\ \Phi & \equiv & \forall n{:}\mathbf{node}\ \forall m{:}\mathbf{node}\quad \mathbf{edge}(n,m) \Rightarrow \mathbf{edge}(m,n) \end{array}$$

If G is a structure of type $\overline{\Gamma}$ and x is a variable declared in Γ, then we write $G.x$ for the value that G assigns to x. The inference rules for structures will allow us to derive the following.

$$G{:}\mathbf{DiGraph} \vdash (\forall n{:}G.\mathbf{node}\ \forall m{:}G.\mathbf{node}\ \ \mathbf{G.edge}(n,m) \Rightarrow \mathbf{G.edge}(m.n)) : \mathbf{bool}$$

This sequent says that a certain formula involving $G.\mathbf{node}$ and $G.\mathbf{edge}$ is well formed, i.e., is an expression of type **bool**.

We can construct a structure object of type $\overline{\Gamma}$ by specifying a value for each variable declared in Γ. For example we can define the complete graph on a type α as follows.

$$\mathbf{CompleteGraph}[\alpha] \equiv \langle \mathbf{node} \leftarrow \alpha;\ \mathbf{edge} \leftarrow \lambda(n{:}\alpha,\ m{:}\alpha)\ \mathbf{True}\rangle$$

As another example we define the natural number context Γ_N to be the following sequence of variable declarations and assumptions.

$$N{:}\mathbf{type}_1;\ \ 0{:}N;\quad s{:}\ N \to N;$$

$$\neg\exists\ x{:}N\ 0 =_N s(x);$$

$$\forall x{:}N, y{:}N\ \ s(x) =_N s(y) \Rightarrow (x =_N y);$$

$$\begin{array}{l}\forall P{:}(N \to \mathbf{bool}) \\ \quad P(0) \wedge (\forall x{:}N\ P(x) \Rightarrow P(s(x))) \\ \qquad \Rightarrow \forall x{:}N\ P(x)\end{array}$$

We can define the natural numbers to be a structure satisfying these axioms, i.e., a structure $\mathcal{N}$ such that $\vdash \mathcal{N}{:}\overline{\Gamma_N}$

4.1 Forming Structure Types

The inference rules for structure type formation are given in figure 4.1. We first focus on the empty structure rule and structure type formation rules A and B. These antecedents are important only in the presence of the constructs introduced in the appendices. The more complex rule of Structure Type Formation C is discussed below.

The rules of structure type formation A and B mirror corresponding rules for constructing well formed contexts. These rules can be used to construct the following (abbreviated) derivation.

1	$\vdash$	$\overline{\varepsilon}{:}\mathbf{type}_0$	empty structure
2	$\vdash$	$\overline{\varepsilon}{:}\mathbf{type}_1$	type promotion from 1
3	$\vdash$	$\overline{\mathbf{node}{:}type_0}{:}\mathbf{type}_1$	struct. type form. A from 2
4	$\vdash$	$\overline{\mathbf{node}{:}type_0; edge{:}\mathbf{node}\times\mathbf{node}\to\mathbf{bool}}{:}\mathbf{type}_1$	struct. type form. A from 3
5	$\vdash$	$\overline{\mathbf{node}{:}type_0; edge{:}\mathbf{node}\times\mathbf{node}\to\mathbf{bool}; \Phi{:}\mathbf{type}_1}$	struct. type form. B from 4

Problem 4.1: The natural number context Γ_N can written be as follows where A_N is a Boolean expression stating the axioms of the natural numbers.

$$N{:}\mathbf{type}_1;\ 0{:}N;\ S{:}N \to N;\ A_N$$

Give a derivation of $\vdash \overline{\Gamma}_N{:}\mathbf{type}_2$.

We now consider the rule of structure type formation C. The structure types that have been considered so far are *closed* in the sense that they do not contain free variables. In particular, if $\Gamma \vdash \mathbf{True}$ then every variable occurring in Γ must be declared in Γ and this implies that there are no free variables in the type $\overline{\Gamma}$. This should be contrasted with a structure type such as $\mathbf{PairOf}[\alpha, \beta]$ which is type of pairs whose first component is an instance of the α and whose second component is an instance of β. Here α and β are type variables and

Structure Type Formation A:

$$\frac{}{\vdash \overline{\varepsilon} : \mathbf{type}_0}$$

Structure Type Formation B:

$$\frac{\vdash \overline{\Gamma} : \mathbf{type}_i \quad \Gamma \vdash \sigma : \mathbf{type}_i \quad x \text{ is not declared in } \Gamma}{\vdash \overline{\Gamma; x:\sigma} : \mathbf{type}_i}$$

Structure Type Formation C:

$$\frac{\vdash \overline{\Gamma} : \mathbf{type}_i \quad \Gamma \vdash \Phi : \mathbf{bool}}{\vdash \overline{\Gamma; \Phi} : \mathbf{type}_i}$$

Structure Type Formation D:

$$\frac{\vdash \overline{\Delta; \Gamma} : \mathbf{type}_i \quad \Sigma \vdash D : \overline{\Delta}}{\Sigma \vdash \overline{\Gamma}_{\Delta}(D) : \mathbf{type}_i}$$

Figure 4.1: **Structure type formation.**

the meaning of the type $\mathbf{PairOf}[\alpha, \beta]$ depends on the meaning assigned to the free type variables α and β. There are no free variables of the type **Graph** — the meaning of the type **Graph** does not depend on the meaning of any free type variables. The rule of structure type formation C allows us to construct structure types such as $\mathbf{PairOf}[\alpha, \beta]$ which contain free type variables. For any type expressions σ and τ we will use the rule of type formation C to construct a representation of the type $\mathbf{PairOf}[\sigma, \tau]$. We first consider the following context.

$$\alpha : \mathbf{type}_1;\ \beta : \mathbf{type}_1;\ \mathbf{first} : \alpha;\ \mathbf{second} : \beta$$

We can write this context as follows.

$$\Delta; \Gamma$$

$$\begin{array}{rcl} \Delta & \equiv & \alpha : \mathbf{type}_1;\ \beta : \mathbf{type}_1 \\ \Gamma & \equiv & \mathbf{first} : \alpha;\ \mathbf{second} : \beta \end{array}$$

Using the inference rules of structure formation A and B we can derive the following for these particular Δ and Γ

$$\vdash \overline{\Delta; \Gamma} : \mathbf{type}_2$$

Now we will consider a context Σ and expressions σ and τ such that we have the following sequents.

$$\Sigma \vdash \sigma : \mathbf{type}_1$$

$$\Sigma \vdash \tau : \mathbf{type}_1$$

Using the inference rules of structure formation in figure 4.2, discussed below, we can derive the following sequent.

$$\Sigma \vdash \langle \alpha \leftarrow \sigma,\ \beta \leftarrow \tau \rangle : \overline{\Delta}$$

Using the inference rule of structure type formation C we can then derive the following sequent.

$$\Sigma \vdash \overline{\mathbf{first}:\alpha;\ \mathbf{second}:\beta}\ _{\Delta}(\langle \alpha \leftarrow \sigma,\ \beta \leftarrow \tau \rangle) : \mathbf{type}_2$$

The structure type $\overline{\mathbf{first}:\alpha;\ \mathbf{second}:\beta}\ _{\Delta}(\langle \alpha \leftarrow \sigma,\ \beta \leftarrow \tau \rangle)$ is our representation of the type $\mathbf{PairOf}[\sigma, \tau]$. For a general structure type of the form $\overline{\Gamma}_{\Delta}(D)$ we can think of D as a substitution — it assigns values to the free variables of Γ. One might think that we could simply apply the substitution and write, for example, the type $\mathbf{PairOf}[\sigma, \tau]$ as $\overline{\mathbf{first}:\sigma;\ \mathbf{second}:\tau}$. But this is problematic if τ contains **first** as a free variable. In general we cannot apply the substitution implicit in the type $\overline{\Gamma}_{\Delta}(D)$ because we cannot rename the bound variables of Γ — the structure slot names — to avoid the capture of free variables in D.

As another example we can consider multisets (also known as bags). A multiset (or bag) is a set in which a given item can occur more than once. We can formalize the type $\mathbf{BagOf}[\sigma]$ as follows.

$$\mathbf{BagOf}[\sigma] \equiv \overline{I:\mathbf{type}_0;\ f:I \to \alpha}\ _{\alpha:\mathbf{type}_1}\ (\langle \alpha \leftarrow \sigma \rangle)$$

Chapter 5 will show that two bags of type $\mathbf{BagOf}[\sigma]$ are equal (isomorphic) if for each x of type σ the two bags contain x the same number of times. The type $\mathbf{BagOf}[\sigma]$ is perhaps the simplest nontrivial structure type involving both a free type variable and a bound type variable.

Another example similar to bags is sequences. The type $\mathbf{SequenceOf}[\sigma]$ can be defined as follows.

$$\mathbf{SequenceOf}[\sigma] \equiv \overline{I:\mathbf{type}_0;\ \leq:\mathbf{TotalOrderOn}[I];\ f:I \to \alpha}\ _{\alpha:\mathbf{type}_1}\ (\langle \alpha \leftarrow \sigma \rangle)$$

Here the type $\mathbf{TotalOrderOn}[\sigma]$ is the type $\mathbf{TheSubtype}(R{:}\sigma{\times}\sigma \to \mathbf{bool} \mid \mathcal{A})$ where $\mathcal{A}$ is a Boolean expression stating that R is a total order on σ. Two sequences are equal if there is a bijection between their index sets which preserves both $\leq$ and f.

4.2 Forming Structures

For the rules in figure 4.2 we adopt the convention that $\overline{\Gamma}$ is actually an abbreviation for $\overline{\Gamma}_{\varepsilon}(\langle\rangle)$. Under this convention we get the following special case of the structure variable rule by taking Δ to be ε and D to be $\langle\rangle$.

Structure Variable:

$$\frac{\Sigma \vdash G : \overline{\Gamma}_\Delta(D) \qquad \Delta;\Gamma \vdash \Theta}{\Sigma \vdash \mathcal{V}_{\Delta;\Gamma}\,[\![\Theta]\!]\,D;G}$$

Structure Instance Formation:

$$\frac{\vdash \overline{\Delta;\Gamma} : \mathbf{type}_i \qquad \Sigma \vdash \mathcal{V}_{\Delta;\Gamma}\,[\![\Theta]\!]\,D;G \quad \forall\,\Theta \in \Delta;\Gamma}{\Sigma \vdash G : \overline{\Gamma}_\Delta(D)}$$

Structure Slot Reference:

$$\frac{\Sigma \vdash e_1 : \tau_1 \;\cdots\; \Sigma \vdash e_n : \tau_n}{\Sigma \vdash \langle x_1 \leftarrow e_1; \cdots \;\; x_n \leftarrow e_n \rangle\,.x_i = e_i}$$

Figure 4.2: **Structure variable and instance formation rules.** The structure type $\overline{\Gamma}$ is here taken to be an abbreviation for $\overline{\Gamma}_\varepsilon(\langle\rangle)$ so that we need only consider structure types of the general form $\overline{\Gamma}_\Delta(D)$. See the text for an explanation of the notation $\mathcal{V}_{\Delta;\Gamma}\,[\![\Theta]\!]\,D;G$.

Simple Structure Variable:

$$\frac{\Sigma \vdash G : \overline{\Gamma} \qquad \Gamma \vdash \Theta}{\Sigma \vdash \mathcal{V}_\Gamma\,[\![\Theta]\!]\,G}$$

In this rule G may be a variable declared in Σ to be of type $\overline{\Gamma}$, hence the name "variable rule". For G a structure of type $\overline{\Gamma}$ we define $\mathcal{V}_\Gamma\,[\![\Theta]\!]\,G$, which we read as "the value of Θ in the structure G", to be the following substitution where $z_1, \ldots, z_n$ are all variables declared in Γ.

$$\mathcal{V}_\Gamma\,[\![\Theta]\!]\,G \equiv \Theta[z_1 \leftarrow G.z_1, \ldots, z_n \leftarrow G.z_n]$$

For example let Γ be the context $\mathbf{node} : \mathbf{type}_0;\ \mathbf{edge} : \mathbf{node} \times \mathbf{node} \to \mathbf{bool}$. For $G : \overline{\Gamma}$ we have the following.

$$\mathcal{V}_\Gamma\,[\![\exists\, x : \mathbf{node}\ \mathbf{edge}(x,x)]\!]\,G \quad\equiv\quad \exists\, x : G.\mathbf{node}\ \ G.\mathbf{edge}(x,x)$$

The simple structure variable rule can be read as saying that if Γ implies Θ, and G is a model of Γ in the sense that G assigns values to variables a way that satisfies Γ, then Θ is true in G in the sense that Θ is true under the variable values assigned by G.

As an example consider the natural number context Γ_N defined at the beginning of the chapter. We can instantiate the simple structure variable rule with the particular natural number context Γ_N to get the following rule.

Natural Number Rule:

$$\frac{\Gamma_N \vdash \Theta \quad \Sigma \vdash \mathcal{N} : \overline{\Gamma_N}}{\Sigma \vdash \mathcal{V}_{\Gamma_N}[\![\Theta]\!]\,\mathcal{N}}$$

In this rule we have that $\mathcal{N}$ is an implementation of the natural numbers — it specifies a type of numbers and a meaning for the constant 0 and the successor function s. The rule says that any statement that is provable from the context Γ_N, i.e., provable from the axioms of arithmetic, is true for the particular implementation $\mathcal{N}$.

In the general structure variable rule in figure 4.1 we use the following abbreviation where u_1, ..., u_n are the variables declared in Δ and z_1, ..., z_m are the variables declared in Γ.

$$\mathcal{V}_{\Delta;\Gamma}[\![\Theta]\!]\,D; G \equiv \Theta[u_1 \leftarrow D.u_1; , \ldots u_n \leftarrow D.u_n; z_1 \leftarrow G.z_1; , \ldots, z_n \leftarrow G.z_m]$$

Now we consider the structure formation rule. If we take Δ to be the empty context then the structure formation rule reduces to the following somewhat simpler rule our convention that $\overline{\Gamma}$ is an abbreviation for $\overline{\Gamma}_\varepsilon(\langle\rangle)$.

Simple Structure Formation:

$$\frac{\vdash \overline{\Gamma} : \mathbf{type}_i \quad \Gamma \text{ declares } y_1, \ldots, y_m \quad G \text{ is } \langle y_1 \leftarrow s_1; \cdots ; y_m \leftarrow s_m \rangle \quad \Sigma \vdash \mathcal{V}_\Gamma[\![\Theta]\!]\,G \ \ \forall\, \Theta \in \Gamma}{\Sigma \vdash G : \overline{\Gamma}}$$

The last antecedent of this structure formation rule abbreviates a separate antecedent for each declaration and assumption in Γ. A use of this rule will typically have a large number of antecedents.

Problem 4.2: Give a derivation using the simple structure formation rule of the following sequent.

$$\alpha : \mathbf{type}_0 \vdash \mathbf{CompleteGraph}[\alpha] : \mathbf{graph}$$

The general structure formation rule is a straightforward generalization of the simple structure formation rule. The slot reference rules derives an absolute equality and should be self explanatory.

Problem 4.3: Let Σ be the context $\alpha : \mathbf{type}_1$; $f : \alpha \to \alpha$ and let M be an expression such that $\vdash M : \overline{\Sigma}$. Give the result of the subsitution $\mathcal{V}_\Sigma[\![\forall x{:}\alpha \ \ f(x) =_\alpha x]\!]\,M$.

Chapter 5

Isomorphism

Equality at structure types is called isomorphism. As an example of isomorphism consider the type of finite directed graphs which we have defined as follows.

$$\mathbf{DiGraph} \equiv \overline{\mathbf{node}{:}\mathbf{type}_0;\ \mathbf{edge}{:}\ (\mathbf{node} \times \mathbf{node}) \to \mathbf{bool}}$$

Two directed graphs G and H are isomorphic if there exists a bijection f from the nodes of G — the instances of the type $G.\mathbf{node}$ — to the nodes of H — the instances of the type $H.\mathbf{nodes}$ — such that for any two nodes n and m of G we have that there is an edge from $f(n)$ to $f(m)$ in H if and only if there exists an edge from n to m in G.

As another example we can consider hypergraphs. A hypergraph consists of a type specifying the nodes together with a set of hyperedges where each hyperedge is itself a set of nodes (rather than a pair of nodes). In Boolean type theory sets are represented by predicates. Hence the type of a hyperedge is $\mathbf{node} \to \mathbf{bool}$ and a set of hypedges is represented by a predicate on hyperedges — a set of hypedges has type $(\mathbf{node} \to \mathbf{bool}) \to \mathbf{bool}$. The type of finite hypergraphs can be represented as follows.

$$\mathbf{HyperGraph} \equiv \overline{\mathbf{node}{:}\mathbf{type}_0;\ \mathbf{edge}{:}\ (\mathbf{node} \to \mathbf{bool}) \to \mathbf{bool}}$$

It is useful to compare the structure type for a hypergraph with the structure type for a directed graph.

Two hypergraphs G and H are isomorphic if there exists a bijection f from the nodes of G to the nodes of H such that for any predicate P on the nodes of G we have that P is an edge of G if and only if the corresponding predicate on the nodes of H, as defined by the bijection f, is a an edge of H. If P is a predicate on the nodes of G (defining a subset of the nodes of G) the corresponding predicate P' on the nodes of H is defined by the following condition.

$$\forall\, x{:}G.\mathbf{node}\ \ P'(f(x)) \Leftrightarrow P(x)$$

As a third example, consider the following structure type which we will call

a successor structure.

$$\mathbf{SStruct} \equiv \overline{\mathbf{member}:\mathbf{type}_1;\ s:\mathbf{member} \to \mathbf{member}}$$

If M is a successor structure then $M.\mathbf{member}$ is a type and the instances of $M.\mathbf{member}$ will be called the members of M. In this case we allow an infinite number of members. Here we have that two successor structures M ad N are isomorphic if there exists a bijection f from the members of M to the members of N such that for any member x of M we have that $f(M.s(x)) =_{H.\mathbf{member}} N.s(f(x))$. This can be phrased as saying that the input-output pairs of the function $M.s$ correspond, under the identification defined by the bijection f, to the input-output pairs of the function $N.s$.

As a fourth example we consider "magmas" which can be defined as follows.

$$\mathbf{Magma} \equiv \overline{\mathbf{member}:\mathbf{type}_1;\ h:\mathbf{member} \times \mathbf{member} \to \mathbf{member}}$$

Two magmas M and N are isomorphic if there exists a bijection f from the members of M to the members of N such that for any members x and y of M we have that $f(M.h(x,y)) = N.h(f(x),f(y))$. This can again be phrased as saying that the input-output triples of the function $M.h$ correspond, under the identification defined by the bijection f, to the input-output triples of the function $N.h$.

5.1 Isomorphism for Simple Structure Types

A type expression is simple over a given set of type variables if it is either the type **bool**, one of the given type variables, or a function type $\sigma \to \tau$ where σ and τ are recursively simple. Equivalently, we can define a simple type expression by the following grammar where α ranges over the given set of type variables.

$$\tau ::= \alpha \mid \mathbf{bool} \mid \tau \to \tau$$

Note that $\sigma \times \gamma \to \tau$ is an abbreviation for $\sigma \to (\gamma \to \tau)$ and is therefore a simple type expression provided that σ, γ, and τ are simple.

A context Γ will be called simple if every variable declaration in Γ is either a type variable declaration of the form $\alpha : \mathbf{type}_i$ or has the form $x : \tau$ where τ is a simple type expression over the type variables declared earlier in Γ. For example the following context is simple.

$$\alpha:\mathbf{type}_1;\ \beta:\mathbf{type}_1;\ f:(\alpha \to \beta) \to (\beta \to \alpha)$$

A structure type is simple if it is of the form $\overline{\Gamma}$ where Γ is a simple context.

Consider a simple context Γ and a structure $M:\overline{\Gamma}$. The structure M assigns a meaning to each of the variables declared in Γ and, in particular, it assigns a meaning to the type variables declared in Γ. Given a meaning for the type

variables we can assign a meaning to each simple type expression defined by the following equations.

$$\begin{aligned}\mathcal{V}_\Gamma\,[\![\alpha]\!]\,M &= M.\alpha\\ \mathcal{V}_\Gamma\,[\![\mathbf{bool}]\!]\,M &= \mathbf{bool}\\ \mathcal{V}_\Gamma\,[\![\sigma\to\tau]\!]\,M &= \mathcal{V}_\Gamma\,[\![\sigma]\!]\,M \to \mathcal{V}_\Gamma\,[\![\tau]\!]\,M\end{aligned}$$

Recall that we have define $\mathcal{V}_\Gamma\,[\![e]\!]\,M$ to be the result of substituting $M.x$ for each variable x declared in Γ. The above equations follow from the nature of syntactic substitution. However, we can also think of them as equations about the "meaning" of the type expressions under the interpretations of the type variables specified by M.

Now consider a simple context Γ and two structures $M:\overline{\Gamma}$ and $N:\overline{\Gamma}$. Suppose that for each type variable α declared in Γ we are given a bijection f_α between the types $M.\alpha$ and $N.\alpha$. We will call this a system of type variable correspondences between the structures M and N. Given such a system, and any simple type τ over the type variables declared in Γ, we now define a bijection f_γ from the types $\mathcal{V}_\Gamma\,[\![\tau]\!]\,M$ to the type $\mathcal{V}_\Gamma\,[\![\tau]\!]\,N$. These bijections are define by recursion on type expressions using the following equations.

$$\forall\, P{:}\mathbf{bool} \qquad f_{\mathbf{bool}}(P) = P$$

$$\forall\, x{:}M.\alpha \qquad f_\alpha(x) \qquad \text{is given for } \alpha \text{ declared n } \Gamma$$

$$\forall\, h{:}\mathcal{V}_\Gamma\,[\![\sigma\to\tau]\!]\,M \qquad f_{\sigma\to\tau}(h) = \mathbf{The}\left(\begin{array}{c} g{:}\mathcal{V}_\Gamma\,[\![\sigma\to\tau]\!]\,N\\ \forall x{:}\mathcal{V}_\Gamma\,[\![\sigma]\!]\,M\\ g(f_\sigma(x)) =_{\mathcal{V}_\Gamma[\![\tau]\!]N} f_\tau(h(x))\end{array}\right)$$

The last equation defining $f_{\sigma\to\tau}(h)$ can be drawn as the following "commutative diagram".

$$\begin{array}{ccc}\mathcal{V}_\Gamma\,[\![\sigma]\!]\,M & \overset{h}{\to} & \mathcal{V}_\Gamma\,[\![\tau]\!]\,M\\ f_\sigma\downarrow & & \downarrow f_\tau\\ \mathcal{V}_\Gamma\,[\![\sigma]\!]\,N & \overset{g}{\to} & \mathcal{V}_\Gamma\,[\![\tau]\!]\,N\end{array}$$

Because f_σ is a bijection, and hence invertible, this diagram defines g. Conversely, because f_τ is invertible g determines h. So the function $f_{\sigma\to\tau}$ which maps h to g is also a bijection as desired.

For simple structure types we now define $M =_{\overline{\Gamma}} N$ to mean that there exists a system of type variable correspondences between M and N such that for all declarations $h{:}\tau$ in Γ with τ a function type we have $N.h =_{\mathcal{V}_\Gamma[\![\tau]\!]N} f_\tau(M.h)$.

Problem 5.1: Consider the contest $\Sigma \equiv \alpha : \mathbf{type}_1;\ \bullet : \alpha \times \alpha \to \alpha$ and consider two structures $N, N : \overline{\Sigma}$ such that there exists a bijection f_α from $N.\alpha$ to $M.\alpha$ such that for $x, y : N.\alpha$ we have $f_\alpha(x(N.\bullet)y) = f_\alpha(x)(M.\bullet)f_\alpha(y)$. Use the fact that $\alpha \times \alpha \to \alpha$ abbreviates $\alpha \to (\alpha \to \alpha)$ to show that $N =_{\overline{\Sigma}} M$. More specifically, show that $f_{\alpha\to(\alpha\to\alpha)}(N.f) =_{M.\alpha\to(M.\alpha\to M.\alpha)} M.f$.

5.2 Isomorphism for Semi-Simple Structure Types

Section 4.1 defined the type $\mathbf{BagOf}[\sigma]$ as follows.

$$\mathbf{BagOf}[\sigma] \equiv \overline{I:\mathbf{type}_0;\ h:I\to\alpha}\ _{\alpha:\mathbf{type}_1}\ (\langle \alpha \leftarrow \sigma \rangle)$$

This is an instance of the general structure type $\overline{\Gamma}_\Delta(D)$. Types of this general form must satisfy the condition that $\Delta;\Gamma$ is a well-formed context and that D is a structure of type $\overline{\Delta}$. A semi-simple structure type is a type of the form $\overline{\Gamma}_\Delta(D)$ where the context $\Delta;\Gamma$ is simple. Note that this definition places no restrictions on the complexity of D. For any well-formed (arbitrarily complex) type expression σ, the type $\mathbf{BagOf}[\sigma]$ is a semi-simple type expression.

A fundamental property of type-relative equality is that objects which are equal at type τ cannot be distinguished by well-formed predicates of type $\tau \to \mathbf{bool}$. We will now consider the question of when two objects of type $\mathbf{BagOf}[\sigma]$ are distinguishable.

For a bag $B:\mathbf{BagOf}[\sigma]$ we will call instances of the type $B.I$ "indices" and for $x:\sigma$ we define the count of x in B to be the number of indices $i:(B.I)$ such that $(B.h)(i) =_\sigma x$. We say that two bags $B:\mathbf{BagOf}[\sigma]$ and $C:\mathbf{BagOf}[\sigma]$ define the same counts if for every $x:\sigma$ we have that the count of x in B equals the count of x in C. We will first show that if bags $B:\mathbf{BagOf}[\sigma]$ and $C:\mathbf{BagOf}[\sigma]$ define different counts on the elements of σ then B and C can be distinguished by well formed predicates on $\mathbf{BagOf}[\sigma]$.

To show that bags defining different counts can be distinguished, consider the following predicate on the type $\mathbf{BagOf}[\sigma]$ where x is an instance of σ.

$$\lambda\, M:\mathbf{BagOf}[\sigma] \quad \exists i, j : M.I \quad i \neq_{M.I} j \quad \wedge \quad (M.h)(i) =_\sigma x \wedge (M.h)(j) =_\sigma x$$

This predicate is true of a bag M if and only if M contains x at least twice. There is one such predicate for each instance x of σ. Hence if there exists an instance x of σ such that B contains x at least twice but C does not contain x at least twice then B and C can be distinguished. A similar predicate can be constructed for containing x at least n times for any n. Hence if any count is different on any instance of σ the bags can be distinguished.

We now claim that if $B:\mathbf{BagOf}[\sigma]$ and $C:\mathbf{BagOf}[\sigma]$ define the same counts on the elements of σ then they cannot be distinguished by well formed predicates on $\mathbf{BagOf}[\sigma]$. For an instance M of the type $\mathbf{BagOf}[\sigma]$ the choice of the index type $M.I$ is arbitrary. Two bags A and B of type $\mathbf{BagOf}[\sigma]$ are isomorphic as instances of $\mathbf{BagOf}[\sigma]$ if there exists a bijection f from the index type $A.I$ to

the index type $B.I$ such for any index $i : A.I$ we have $(B.h)(f(i)) =_\sigma (A.h)(i)$. It is not difficult to show that this definition is equivalent to the statement that A and B define the same counts.

We now consider semi-simple structure types in general. A semi-simple structure type has the form $\overline{\Gamma}_\Delta(D)$ where $\Delta;\Gamma$ is a simple context. We consider the simple types that can be constructed from the type variables declared in both Δ and Γ. An instance G of type $\overline{\Gamma}_\Delta(D)$ assigns a meaning to each such simple type as defined by the following equations.

$$\begin{array}{rcl}
\mathcal{V}_{\Delta;\Gamma}[\![\beta]\!]\, D; G & = & D.\beta \quad \text{for } \beta \text{ declared in } \Delta \\
\mathcal{V}_{\Delta;\Gamma}[\![\alpha]\!]\, D; G & = & G.\alpha \quad \text{for } \alpha \text{ declared in } \Gamma \\
\mathcal{V}_{\Delta;\Gamma}[\![\mathbf{bool}]\!]\, D; G & = & \mathbf{bool} \\
\mathcal{V}_{\Delta;\Gamma}[\![\sigma \to \tau]\!]\, D; G & = & \mathcal{V}_{\Delta,\Gamma}[\![\sigma]\!]\, D; G \to \mathcal{V}_{\Delta;\Gamma}[\![\tau]\!]\, D; G
\end{array}$$

Consider two instances M ad N of type $\overline{\Gamma}_\Delta(D)$. We define a system of correspondences between M and N to consist of a bijection between the types $M.\alpha$ and $N.\alpha$ for each type variable α with $\alpha : \mathbf{type}_i$ in Γ. We do not assign correspondences for the type variables declared in Δ. Given a system of correspondences from an instance $M : \overline{\Gamma}_\Delta(D)$ to $N : \overline{\Gamma}_\Delta(D)$, and a simple type τ over the type variables declared in Δ and Γ, we can define a bijection f_τ from $\mathcal{V}_{\Delta;\Gamma}[\![\tau]\!]\, D; M$ to $\mathcal{V}_{\Delta;\Gamma}[\![\tau]\!]\, D; M$ by the following equations.

$$\begin{array}{lrcl}
\forall\, P : \mathbf{bool} & f_{\mathbf{bool}}(P) & = & P \\
\forall\, x : D.\beta & f_\beta(x) & =_{D.\beta} & x \quad \text{for } \beta : \mathbf{type}_i \text{ in } \Delta \\
\forall\, x : M.\alpha & f_\alpha(x) & & \text{is given for } \alpha : \mathbf{type}_i \text{ in } \Gamma \\
\forall\, h : \mathcal{V}_\Gamma[\![\sigma \to \tau]\!]\, M & f_{\sigma\to\tau}(h) & = & \mathbf{The} \left(\begin{array}{l} g : \mathcal{V}_\Gamma[\![\sigma \to \tau]\!]\, N \\ \quad \forall x : \mathcal{V}_\Gamma[\![\sigma]\!]\, M \\ \quad\quad g(f_\sigma(x)) =_{\mathcal{V}_\Gamma[\![\tau]\!]N} \quad f_\tau(h(x)) \end{array} \right)
\end{array}$$

In the equations, as in the equations for simple structure types, the bijection f_τ is indexed by the type expression τ rather than the meaning of τ in any particular structure.

Two structures $M : \overline{\Gamma}_\Delta(D)$ and $M : \overline{\Gamma}_\Delta(D)$ are isomorphic as instances of $\overline{\Gamma}_\Delta(D)$ if there exists a system of bijections from M to N such that for all declaration $h : \tau$ in Γ with τ a function type we have $N.h =_{\mathcal{V}_{\Delta;\Gamma}[\![\tau]\!]D;N} \ f_\tau(M.h)$.

Problem 5.2: Consider the structure type $\mathbf{PairOf}[\sigma, \tau]$ which was defined section 4.1 as follows.

$$\mathbf{PairOf}[\sigma,\tau] \equiv \overline{\mathbf{first} : \alpha;\ \mathbf{second} : \beta}_{\,\Delta}(\langle \alpha \leftarrow \sigma,\ \beta \leftarrow \tau \rangle)$$

$$\Delta \equiv \alpha : \mathbf{type}_1;\ \beta : \mathbf{type}_1$$

This is a semi-simple structure type $\overline{\Gamma}_\Delta(D)$ where Γ does not declare any type variables. Now consider $p : \mathbf{PairOf}[\sigma, \tau]$ and $q : \mathbf{PairOf}[\sigma, \tau]$. Using the general definition of equivalence at semi-simple structure types show that $p =_{\mathbf{PairOf}[\sigma,\tau]} q$ if and only if $p.\mathbf{first} =_\sigma q.\mathbf{first}$ and $p.\mathbf{second} =_\tau q.\mathbf{second}$.

5.3 General Isomorphism

In general structures are first class objects — they can themselves be contained in the slots of structures, be passed as arguments to functions and returned as values from functions. For example, we might define a colored graph as follows.

$$\mathbf{ColoredGraph} \equiv \overline{G{:}\mathbf{Graph};\ \mathbf{color}{:}\mathbf{type}_i;\ c{:}(G.\mathbf{node} \to \mathbf{color})}$$

Two colored graphs M_1 and M_2 are isomorphic if there exists a bijection f_N betwen the nodes of $M_1.G$ and the nodes of $M_2.G$ and also a bijection f_C between the colors $M_1.C$ and $M_2.C$ such that these two bijections identify M_1 and M_2. More formally the bijections must be such that for every pair of nodes n and m of $M_1.G$ we have that there exists an edge from n to m in $M_1.G$ if and only if there exists an edge from $f_N(n)$ to $f_N(m)$ in $M_2.G$ and, also, for every node n of $M_1.G$ we have that $f_C(M_1.c(n)) =_{M_2.\mathbf{color}} M_2.c(f_N(n))$. In many cases, such as the above colored graph example, the notion of isomorphism seems intuitively clear.

Problem 5.3: A hypergraph is a finite type (set) of "nodes" together with a set of hyper-edges where each hyper-edge is a subset of the set of nodes. Give a structure type for the type "hypergraph".

Problem 5.4: For two hypergraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ give a Boolean expression expressing the condition that $\mathcal{G}_1$ is isomorphic to $\mathcal{G}_2$.

Chapter 6

Voldemort's Theorem

Voldemort's theorem states that certain objects cannot be named. If a circle is defined in a coordinate-free way, there is no way to name a point on the circle — any two points satisfy exactly the same coordinate-free properties. A similar statements holds for the four corners of a square, or the points on a coordinate-free (affine) plane. Voldemort's theorem says that if there exists a symmetry carrying x to y then no property can name x (without also naming y). While this may seem obvious, it is perhaps less obvious that the concept of symmetry can be defined for all mathematical objects — we can associate a notion of symmetry with every mathematical type.

Consider the directed graph G_1 shown below.

$$G_1: \quad \begin{array}{ccc} A & \rightarrow & B \\ \uparrow & & \downarrow \\ D & \leftarrow & C \end{array}$$

This graph is clearly isomorphic to the directed graph G_2 shown below.

$$G_2: \quad \begin{array}{ccc} X & \rightarrow & Y \\ \uparrow & & \downarrow \\ W & \leftarrow & Z \end{array}$$

However, the isomorphism is not unique. We can identify the node A with any of the nodes X, Y, Z or W. For example, if A goes to Z then B goes to W, C goes to X and D goes to Y.

Although any structure is isomorphic to itself, there can exist different isomorphisms between an object and itself. An isomorphism between an object and itself is called a *symmetry* of that object. The graph G_1 has four symmetries — we can construct an isomorphisms of G_1 with itself by mapping node A to any one of the four nodes and then mapping the remaining nodes around the loop. For example, we can map A to C, B to D, C to A and D to B.

An isomorphism between two objects is a system of bijections between the types of those objects. An isomorphism of an object with itself is a system of

bijections from types to themselves — it is a system of permutations. Like the notion of isomorphism, the notion of symmetry is fundamental to mathematics in general.

In developing a general notion of symmetry it will be useful to use the following "dependent pair" type $\mathbf{PairOf}[\sigma, \tau[\cdot]]$ which generalizes the type $\mathbf{PairOf}[\sigma, \tau]$ defined in section 4.1.

$$\begin{array}{rcl} \mathbf{PairOf}[\sigma, \tau[\cdot]] & \equiv & \overline{\mathbf{First}:\alpha;\ \mathbf{Second}:F(\mathbf{First})}_{\Delta}(D) \\ \Delta & \equiv & \alpha:\mathbf{type}_1; F:\alpha \to \mathbf{type}_1 \\ D & \equiv & \langle \alpha \leftarrow \sigma;\ F \leftarrow \lambda\, M:\sigma\ \tau[M] \rangle \end{array}$$

Here we will write $\langle G, x \rangle$ as an abbreviation for $\langle \mathbf{First} \leftarrow G;\ \mathbf{Second} \leftarrow x \rangle$. For $G:\sigma$ and $x:\tau[G]$ we have that $\langle G, x \rangle : \mathbf{PairOf}[\sigma, \tau[\cdot]]$. We have

$$\langle G, x \rangle =_{\mathbf{PairOf}[\sigma,\tau[\cdot]]} \langle G, y \rangle$$

if there is σ-symmetry of G which yields an identification of x and y (we say that symmetry "carries" x to y).

In the symmetric graph G_1 above we cannot name any particular node of the graph using graph-theoretic properties. We say that the type $G_1.\alpha$ has no canonical instance. More generally we define the non-existence of a canonical instance of a type $\tau[G]$ as follows.

Definition 1. *We say that there is no canonical instance of a type $\tau[G]$ for $G:\sigma$ in context Σ if $\Sigma \vdash \sigma:\mathbf{type}_1$, $\Sigma; M:\sigma \vdash \tau[M]:\mathbf{type}_1$, $\Sigma \vdash G:\sigma$ and*

$$\begin{array}{rl} & \forall x:\tau[G] \\ \Sigma \vdash & \quad \exists y:\tau[G]\ y \neq_{\tau[G]} x \wedge \\ & \qquad \langle G, y \rangle =_{\mathbf{PairOf}[\sigma,\tau[\cdot]]} \langle G, x \rangle \end{array}$$

Theorem 1 (Voldemort's theorem)**.** *(Assuming the inference rules are consistent ...) If there is no canonical instance of $\tau[G]$ for $G:\sigma$ in context Σ then no instance of $\tau[G]$ can be named by σ-theoretic properties: there is no expression $e[M]$ satisfying $\Sigma;\ M:\sigma \vdash e[M]:\tau[M]$.*

Proof. Suppose $\Sigma; M : \sigma \vdash e[M] : \tau[M]$. Since there is no canonical instance of $\tau[G]$ there must exist $x : \tau[G]$ with $x \neq_{\tau[G]} e[G]$ but $\langle G, x \rangle =_{\mathrm{PairOf}[\sigma,\tau[\cdot]]} \langle G, e[G] \rangle$. Now consider $P \equiv \langle \langle G, e[G] \rangle, e[G] \rangle$ and $Q \equiv \langle \langle G, x \rangle, e[G] \rangle$. We have $P, Q : \mathbf{PairOf}[\mathbf{PairOf}[\sigma, \tau[\cdot]], \tau[\cdot.\mathbf{First}]]$. The objects P and Q are not isomorphic as instances of this type because $P.\mathbf{First}.\mathbf{Second} =_{\tau[G]} P.\mathbf{Second}$ but $Q.\mathbf{First}.\mathbf{Second} \neq_{\tau[G]} Q.\mathbf{Second}$. But we have the following.

$$\Sigma; R:\mathrm{PairOf}[\sigma, \tau[\cdot]] \vdash \langle R, e[R.\mathbf{First}] \rangle : \mathbf{PairOf}[\mathbf{PairOf}[\sigma, \tau[\cdot]], \tau[\cdot.\mathbf{First}]]$$

The rule of substitution of isomorphics allows us to infer that P and Q are isomorphic by substituting $\langle G, e[G] \rangle$ and $\langle G, x \rangle$ respectively for R in the sequent above. We now have a contradiction. □

Part II

Some Mathematics

Chapter 7

Sets and Numbers

Part I of this book develops a formal notion of well-formed expressions and formal derivations (proofs). Part II develops some mathematical content with a focus on concepts relevant to statics and machine learning. We start by constructing the numbers — the integers, rationals, reals and complex numbers. We then superficially introduce groups, rings and fields as important examples of structure types. We spend considerable time on the concept of a vector space and its properties and develop coordinate-free linear algebra and vector calculus. We give a superficial introduction to measure theory (probability theory) and a coordinate-free treatment of the covariance matrix (and second moment) of a probability distribution on a vector space. A variety of other topics are discussed, all with an emphasis on coordinate-free treatments.

In developing some actual mathematical content we convert from formal notation to English. Although definitions and proofs are given in English, the English terms and statements are intended to have clear and unambiguous translations into the formal notation presented in Part I. In some cases the translation into formal notation is explicitly discussed. But in most cases it is left implicit. Even when working with English definitions and statements, however, the formal notation plays a role in determining well-formedness — an English expression is well formed if there exists a straightforward translation into well-formed formal notation. The formal notation is also important in providing an explicit treatment of the concept of isomorphism which occurs in all branches of mathematics.

We start with a discussion of the English use of the term "set" and the relationship between sets and types.

7.1 Sets

The term "set" is ubiquitous in English mathematical texts. As of this writing the Wikipedia page on groups introduces groups with the following statement.

> In mathematics, a group is an algebraic structure consisting of a set together with an operation that combines any two of its elements to form a third element. To qualify as a group, the set and the operation must satisfy ...

This can be written in formal notion as follows where "the set" is the type α, "the operation" is the function $\bullet$, and the group axioms are written together as a (large) Boolean expression $\mathcal{A}$ (the axioms are given explicitly below).

$$\mathbf{Group} \equiv \overline{\alpha : \mathbf{type}_1;\ \bullet : \alpha \times \alpha \to \alpha;\ \mathcal{A}}$$

The term "set" is used in two different ways in English presentations of mathematics. In the above English description the term set refers to any type in $\mathbf{type}_1$. If we say "a foo-space consists of a set F together with ..." then the expression $x \in S$ should be translated as a type declaration $x : F$. Recall that type declarations are not Boolean expressions. For a group variable declared with $G : \mathbf{Group}$ an English expression of the form "if $x \in G$ ..." is ill-formed. Allowing type declarations to be treated as Boolean expressions leads to contradictions — see the discussion at the beginning of chapter 3.

While the types in $\mathbf{type}_1$ are called sets, the types in $\mathbf{type}_2$ are called classes. Classes are too large to be sets. For example, we have $\mathbf{Group} : \mathbf{type}_2$ and we say that the type **Group** is a class. English mathematics almost never discussed $\mathbf{type}_3$, although the type **Category** is too large to be a class and we have $\mathbf{Category} : \mathbf{type}_3$. We will not discuss categories here.
The second use of the term "set" is given in the following statement of the induction principle for the natural number.

> For any set of natural numbers S we have that if S contains zero and is closed under successor — if for $x \in S$ we have $x + 1 \in S$ — then S contains all natural numbers.

The phrase "set of" or "subset of" indicates a predicate rather than a type. The above statement implicitly declares S to be a predicate of type $\mathcal{N} \to \mathbf{bool}$. In this case the expression $x \in S$ represents the predicate application $S(x)$ and is a well-formed Boolean expression provided that we have $x : \mathcal{N}$. In this case we can form statements of the form "If $x \in S$ ...".

While most of the definitions and proofs given in part II of this book are given in English, occasionally it is instructive to also give formal (machine readable) versions of definitions. Formal notation is often more readable if predicates are written in set notation. We will sometimes use the following set-theoretic notation as an alternative to predicate notation.

$$\begin{aligned} \mathbf{SetOf}[\sigma] &\equiv \sigma \to \mathbf{bool} \\ x \in S &\equiv S(x) \\ \{x : \sigma \mid \Phi[x]\} &\equiv \lambda x : \sigma\ \Phi[x] \end{aligned}$$

For a set $S = \{x : \sigma \mid \Phi[x]\}$ we write $\forall x \in S \ \ \Psi[x]$ for $\forall x : \sigma \ \ x \in S \Rightarrow \Psi[x]$. Similarly, $\exists x \in S \ \ \Psi[x]$ abbreviates $\exists x : \sigma \ \ x \in S \wedge \Psi[x]$ and $\Sigma; x \in S \vdash \Theta[x]$ where x is not declared in Σ abbreviates $\Sigma; x : \sigma; x \in S \vdash \Theta[x]$.

7.2 Defining Numbers

The first step in mathematics is to develop the numbers — the natural numbers, the integers, the rationals, the reals, and the complex numbers. The numbers serve as a case study in mathematical construction. There are two approaches to defining particular structures such as the natural numbers or the real numbers — an axiomatic approach and a model approach. An axiomatic approach to the natural numbers is given in the definition of the context Γ_N at the beginning of chapter 4. This context includes axioms for the natural numbers. These axioms have only one model (up to isomorphism) — the structure of the natural numbers. In the model oriented approach developed here we first implement each kind of number. The natural numbers are implemented as the equivalence classes of types in $\mathbf{type}_0$. The integers are implemented as equivalence classes of formal differences of natural numbers. The rationals are implemented as equivalence classes of formal ratios of integers, and the real are implemented as Dedekind cuts in the rationals. In the model-oriented approach, where one first builds an implementation, it is important for formally represent the fact that different implementations are possible and that the choice of implementation is arbitrary. It is possible to formally distinguishing between a particular implementation of the real numbers and the abstract real numbers which are independent of any choice of implementation. We will let $\widetilde{\mathcal{R}}$ denote the implementation of the reals and define the abstract reals $\mathcal{R}$ as follows.

$$\mathcal{R} \equiv \text{The}\left(G{:}\overline{\Gamma}\ \ G =_{\overline{\Gamma}} \widetilde{\mathcal{R}}\right)$$

$$\Gamma \equiv \overline{R{:}\mathbf{type}_1;\ \ +{:}R\times R\to R;\ \times{:}R\times R\to R}$$

Here the implementation $\widetilde{\mathcal{R}}$ serves only to specifying the abstract structure $\mathcal{R}$ up to isomorphism. The naturals, integers and rationals are handled similarly.

7.3 The Natural Numbers

We implement the natural numbers as the isomorphism classes of types in $\mathbf{type}_0$ where addition is defined by disjoint union and multiplication is defined by taking pair types (Cartesian products). We define the implementation structure $\widetilde{\mathcal{N}}$ to be the following.

$$N \leftarrow \mathbf{type}_0$$

$$+ \leftarrow \left(\begin{array}{l} \lambda(\alpha{:}\mathbf{type}_0, \beta{:}\mathbf{type}_0) \\ \quad \mathrm{The} \left(\begin{array}{l} \gamma{:}\mathbf{type}_0 \\ \exists P{:}\gamma \to \mathbf{bool} \\ \quad \mathrm{TheSubtype}(x{:}\gamma \mid P(x)) =_{\mathbf{type}_0} \alpha \\ \quad \wedge\, \mathrm{TheSubtype}(x{:}\gamma \mid \neg P(x)) =_{\mathbf{type}_0} \beta \end{array} \right) \end{array} \right)$$

Multiplication can be defined in terms of Cartesian product as follows (but multiplication need not be included in the structure).

$$\times_{\widetilde{\mathcal{N}}} \equiv \lambda(\alpha{:}\mathbf{type}_0, \beta{:}\mathbf{type}_0)\mathrm{PairOf}(\alpha, \beta)$$

Note that we have $\times_{\widetilde{\mathcal{N}}}{:}\mathbf{type}_0 \times \mathbf{type}_0 \to \mathbf{type}_0$.

The structure $\widetilde{\mathcal{N}}$ is a particular implementation of the natural numbers. To hide the implementation we define the natural number structure $\mathcal{N}$ as follows.

$$\mathcal{N} \equiv \mathrm{The}\left(G{:}\overline{\Gamma}\ \ G =_{\overline{\Gamma}} \widetilde{\mathcal{N}}\right)$$

$$\Gamma \equiv \overline{N{:}\mathbf{type}_1;\ \ +{:}N \times N \to N}$$

Note that we can derive $\widetilde{\mathcal{N}}.N = \mathbf{type}_0$ (with absolute equality) but $\mathcal{N}.N$ can be any type in $\mathbf{type}_1$ that can be placed in one-to-one correspondence with the equivalence classes of $\mathbf{type}_0$.

By abuse of notation we will often use $\mathcal{N}$ as an abbreviation for the type $\mathcal{N}.N$. For example $n{:}\mathcal{N}$ abbreviates $n{:}\mathcal{N}.N$ and $\mathcal{N} \to \mathcal{N}$ abbreviates $\mathcal{N}.N \to \mathcal{N}.N$. Confusion is avoided by the fact that structures and types appear in disjoint contexts.

Although we omit the details here, the isomorphism between $\mathcal{N}$ and $\widetilde{\mathcal{N}}$ is unique. The uniqueness of the isomorphism implies that the multiplication operator $\tilde{\mathcal{N}}$ defines a multiplication operator on $\mathcal{N}$. Also, since the isomorphism is unique, we can define $0_{\mathcal{N}}$ to be the element of $\mathcal{N}$ identified with the isomorphism class of the empty type in $\widetilde{\mathcal{N}}$.

For $\alpha{:}\mathbf{type}_0$ we will write $|\alpha|$ (the cardinality of α) for the natural number corresponding to α under the unique isomorphism between $\mathcal{N}$ and $\widetilde{\mathcal{N}}$.

7.4 The Integers

We implement integers as equivalence classes of formal differences of natural numbers. By "formal difference" we simply mean a pair of natural numbers

(n, m) which we think of as representing the difference $n - m$ where we allow $m > n$.

$$\text{FDiff} \equiv \text{Pairof}(\mathcal{N}, \mathcal{N})$$

We define an equivalence relation on formal differences by noting that $n - m = n' - m'$ if and only if $n + m' = n' + m$. This can be written formally as follows.

$$\sim_{\widetilde{\mathcal{Z}}} \equiv \begin{array}{l} \lambda(x{:}\text{FDiff},\ y{:}\text{FDiff}) \\ \quad x.\text{first}\ +_{\mathcal{N}}\ y.\text{second}\ =_{\mathcal{N}}\ y.\text{first}\ +_{\mathcal{N}}\ x.\text{second} \end{array}$$

We now define a function mapping a formal difference to its equivalence class.

$$\text{ClassOf}_{\widetilde{\mathcal{Z}}} \equiv \lambda x{:}\text{FDiff}\ \ \{y{:}\text{FDiff} \mid y\ \sim_{\widetilde{\mathcal{Z}}} x\}$$

We now define the structure $\widetilde{\mathcal{Z}}$ as follows.

$$Z \ \leftarrow\ \text{TheSubtype}(P{:}\text{FDiff} \to \mathbf{bool} \mid \exists d{:}\text{FDiff}\ \ P\ =_{\text{FDiff}\to\mathbf{bool}}\ \text{ClassOf}_{\widetilde{\mathcal{Z}}}(d))$$

$$+ \ \leftarrow\ \text{The}\left(\begin{array}{l} f{:}Z \times Z \to Z \\ \quad \forall d_1, d_2{:}\text{FDiff} \\ \quad\ f\left(\text{ClassOf}_{\widetilde{\mathcal{Z}}}(d_1), \text{ClassOf}_{\widetilde{\mathcal{Z}}}(d_2)\right) \\ \quad\ =_Z\ \ \text{ClassOf}_{\widetilde{\mathcal{Z}}}\left(\text{Pair}\left(\begin{array}{l} d_1.\text{first}\ +_{\mathcal{N}}\ d_2.\text{first}, \\ d_1.\text{second}\ +_{\mathcal{N}}\ d_2.\text{second} \end{array}\right)\right) \end{array}\right)$$

We can define multiplication on $\widetilde{\mathcal{Z}}$ as follows where, by abuse of notation, we use $\widetilde{\mathcal{Z}}$ as an abbreviation for the type $\widetilde{\mathcal{Z}}.Z$.

$$\times_{\widetilde{\mathcal{Z}}}\ \equiv\ \text{The}\left(\begin{array}{l} f{:}\widetilde{Z} \times \widetilde{Z} \to \widetilde{Z} \\ \quad \forall d_1, d_2{:}\text{FDiff} \\ \quad\ f\left(\text{ClassOf}_{\widetilde{\mathcal{Z}}}(d_1), \text{ClassOf}_{\widetilde{\mathcal{Z}}}(d_2)\right) \\ \quad\ =_Z\ \ \text{ClassOf}_{\widetilde{\mathcal{Z}}}\left(\text{Pair}\left(\begin{array}{l} d_1.\text{first}\ \times_{\mathcal{N}}\ d_2.\text{first} +_{\mathcal{N}} d_1.\text{second}\ \times_{\mathcal{N}}\ d_2.\text{second}, \\ d_1.\text{first}\ \times_{\mathcal{N}}\ d_2.\text{second} +_{\mathcal{N}} d_1.\text{second}\ \times_{\mathcal{N}}\ d_2.\text{first} \end{array}\right)\right) \end{array}\right)$$

Of course other implementations of the integers are possible. As with the natural numbers, we can suppress the implementation by defining the integers $\mathcal{Z}$ to be the structure isomorphic to $\widetilde{\mathcal{Z}}$.

$$\mathcal{Z} \ \equiv\ \text{The}\left(G{:}\overline{\Gamma}\ \ G =_{\overline{\Gamma}} \widetilde{\mathcal{Z}}\right)$$

$$\Gamma \ \equiv\ \overline{Z{:}\mathbf{type}_1;\ \ +{:}Z \times Z \to Z}$$

As with the natural numbers, the isomorphism between $\mathcal{Z}$ and $\widetilde{\mathcal{Z}}$ is unique. Since the isomorphism is unique, multiplication on $\widetilde{\mathcal{Z}}$ defines multiplication on $\mathcal{Z}$. Again, by abuse of notation we will often write $\mathcal{Z}$ as an abbreviation for the type $\mathcal{Z}.Z$.

7.5 The Rationals

We implement rationals as equivalence classes of formal ratios of integers. By "formal ratio" we simply mean a pair of integers (n, m) with $m \neq 0$ where think of this pair as representing the ratio n/m. Such formal ratios are commonly called fractions.

$$\text{Frac} \equiv \text{TheSubtype}\,(p{:}\text{PairOf}\,(\mathcal{Z},\mathcal{Z})\ \mid p.\text{second} \neq_{\mathcal{Z}}\ 0_{\mathcal{Z}})$$

We define an equivalence relation on fractions as follows.

$$\begin{array}{rcl} \sim_{\widetilde{\mathcal{Q}}} & \equiv & \lambda(x{:}\text{Frac},\ y{:}\text{Frac}) \\ & & \quad x.\text{first}\ \times_{\mathcal{Z}}\ y.\text{second}\ =_{\mathcal{Z}}\ y.\text{first}\ \times_{\mathcal{Z}}\ x.\text{second} \end{array}$$

We now define a function mapping a fraction to its equivalence class.

$$\text{ClassOf}_{\widetilde{\mathcal{Q}}}\ \equiv\ \lambda x{:}\text{Frac}\ \ \{y{:}\text{Frac}\mid y\ \sim_{\widetilde{\mathcal{Q}}} x\}$$

We now define the structure $\widetilde{\mathcal{Q}}$ as follows.

$$Q\ \leftarrow\ \text{TheSubtype}(P{:}\text{Frac}\to\mathbf{bool}\mid\exists d{:}\text{Frac}\ \ P\ =_{\text{Frac}\to\mathbf{bool}}\ \text{ClassOf}_{\widetilde{\mathcal{Q}}}(d))$$

$$+\ \leftarrow\ \text{The}\left(\begin{array}{l} f{:}Q\times Q\to Q \\ \quad \forall d_1,d_2{:}\text{Frac} \\ \quad\ f\left(\text{ClassOf}_{\widetilde{\mathcal{Q}}}(d_1),\text{ClassOf}_{\widetilde{\mathcal{Q}}}(d_2)\right) \\ \quad\ =_Q\ \ \text{ClassOf}_{\widetilde{\mathcal{Q}}}\left(\text{Pair}\left(\begin{array}{l} d_1.\text{first}\ \times_{\mathcal{Z}}\ d_2.\text{second} \\ +_{\mathcal{N}}\ d_1.\text{second}\ \times_{\mathcal{Z}}\ d_2.\text{first}, \\ d_1.\text{second}\ \times_{\mathcal{Z}}\ d_2.\text{second}\end{array}\right)\right) \end{array}\right)$$

$$\times\ \leftarrow\ \text{The}\left(\begin{array}{l} f{:}Q\times Q\to Q \\ \quad \forall d_1,d_2{:}\text{Frac} \\ \quad\ f\left(\text{ClassOf}_{\widetilde{\mathcal{Q}}}(d_1),\text{ClassOf}_{\widetilde{\mathcal{Q}}}(d_2)\right) \\ \quad\ =_Q\ \ \text{ClassOf}_{\widetilde{\mathcal{Q}}}\left(\text{Pair}\left(\begin{array}{l} d_1.\text{first}\ \times_{\mathcal{Z}}\ d_2.\text{first}, \\ d_1.\text{second}\ \times_{\mathcal{Z}}\ d_2.\text{second}\end{array}\right)\right) \end{array}\right)$$

Of course other implementations of the rationals are possible. Again we suppress the implementation by defining the rationals $\mathcal{Q}$ to be the structure isomorphic to $\widetilde{\mathcal{Q}}$.

$$\mathcal{Q}\ \equiv\ \text{The}\left(G{:}\overline{\Gamma}\ \ G=_{\overline{\Gamma}}\ \widetilde{\mathcal{Q}}\right)$$

$$\Gamma\ \equiv\ \overline{Q{:}\mathbf{type}_1;\ \ +{:}Q\times Q\to Q;\ \times{:}Q\times Q\to Q}$$

Again the isomorphism between $\mathcal{Q}$ and $\widetilde{\mathcal{Q}}$ is unique. We note, however, that for the rationals we have to include multiplication in the structure type

defining the isomorphism. If we use only addition then the isomorphism is not unique. Any function which multiplies every rational by fixed nonzero rational q defines a "symmetry of scale" for the rationals under addition. A symmetry is an isomorphism of a structure with itself. Without a unique isomorphism between $\mathcal{Q}$ and $\widetilde{\mathcal{Q}}$, a multiplication operator on $\widetilde{\mathcal{Q}}$ would not define a unique multiplication operator on $\mathcal{Q}$.

7.6 The Reals

We implement real numbers as Dedekind cuts. To define Dedekind cuts we first need to define the standard ordering on the natural numbers, integers and rationals. For natural numbers x and y we can define $x \leq y$ to mean that there exists a natural number z such that $x + z = y$. For formal differences of natural numbers (the implementation of integers) we can say that $n - m \leq n' - m'$ if $n + m' \leq n' + m$. For rational numbers we can that $n/m \leq n'/m'$ if $m > 0$ and $m' > 0$ and $nm' \leq n'm$ (every rational has a representation with a positive denominator).

A Dedekind cut is any subset of the rationals which is not empty, does not contain all rationals, does not contain a maximum element, and is downward closed — if it contain q and $q; \leq q$ then it contains q'. A real number x (such as π or $\sqrt{2}$) is represented by the set of rationals $q < x$ which forms a Dedekind cut. Conversely, any Dedekind cut α represents the unique real number which is the infemum of the set of reals x such that $q < x$ for any $q \in \alpha$ (the infemum of the set of upper bounds on α).

The formal definition of a Dedekind cut is given below where a subset of the rationals is represented by a predicate $P : \mathcal{Q} \to \mathbf{bool}$. For a predicate $P : \tau \to \mathbf{bool}$ we will sometimes write a variable declaration $x \in P$ as an abbreviation for $x : \mathrm{TheSubtype}(\tau, P)$. We will sometimes omit subscripts from operators when the intended operator is clear from context.

$$\mathrm{DCut} \;\equiv\; \mathrm{TheSubtype}\left(\begin{array}{l} S : \mathbf{SetOf}[Q] \mid \\ \quad \exists q \in S \\ \quad\; \wedge \;\; \exists q : \mathcal{Q}\; q \notin S \\ \quad\; \wedge \;\; \neg\exists q \in S\; \forall q' \in S\; q' \leq q \\ \quad\; \wedge \;\; \forall q \in P\; \forall q' \in \mathcal{Q}\; q' \leq q \Rightarrow q' \in \mathcal{Q} \end{array}\right)$$

We define addition on Dedekind cuts, $+_{\widehat{\mathcal{R}}} : \mathrm{DCut} \times \mathrm{DCut} \to \mathrm{DCut}$, as follows.

$$\begin{array}{rcl} & & \lambda(P : \mathrm{DCut}, P' : \mathrm{DCut}) \\ +_{\widehat{\mathcal{R}}} & \equiv & \quad \lambda q : \mathcal{Q} \\ & & \quad\; \exists r \in P\; \exists s \in P' \quad q < (r + s) \end{array}$$

Defining multiplication is somewhat more awkward. We start by defining a form of multiplication which is correct if both arguments are non-negative.

$$\times_{\widehat{\mathcal{R}}} \equiv \begin{array}{l}\lambda(P{:}\mathrm{DCut}, P'{:}\mathrm{DCut}) \\ \quad \lambda q{:}\mathcal{Q} \\ \quad\quad \exists r \in P\ \exists s \in P' \quad q < (r \times s)\end{array}$$

To define multiplication for general arguments (possibly negative) we need more terminology. We start by defining zero and the order relation Dedekind cuts.

$$0_{\widetilde{\mathcal{R}}} = \lambda q{:}\mathcal{Q}\ \ q < 0$$

$$\leq_{\widetilde{\mathcal{R}}} = \lambda(P{:}\mathrm{DCut},\ P'{:}\mathrm{DCut})\ \ \forall \quad q \in P\ \ P'(q)$$

Next we define a function which maps a Dedekind cut representing real number x to a Dedekind cut representing $-x$.

$$\mathrm{Negative} \equiv \begin{array}{l}\lambda P{:}\mathrm{DCut} \\ \quad \lambda q{:}\mathcal{Q} \\ \quad\quad \exists r{:}\mathcal{Q} \quad q < r \wedge\ \forall\, w \in P\ \ r < -w\end{array}$$

We now introduce a general notion of conditional expression.

$$\mathrm{If}_\alpha \equiv \begin{array}{l}\lambda(P{:}\mathbf{bool}, x{:}\alpha, y{:}\alpha) \\ \quad \mathrm{The}(z{:}\alpha\ (P \Rightarrow z =_\alpha x) \wedge \neg(P \Rightarrow z =_\alpha y))\end{array}$$

We can then define an absolute value operator as follows.

$$\mathrm{Abs} \equiv \begin{array}{l}\lambda P{:}\mathrm{DCut} \\ \quad \mathrm{If}(\exists q \in P\ \ q > 0, \quad q, \quad \mathrm{Negative}(q))\end{array}$$

Finally we can define the Dedekind cut structure $\widetilde{\mathcal{R}}$ as follows.

$$R \leftarrow \mathrm{DCut}$$

$$+ \leftarrow +_{\widetilde{\mathcal{R}}}$$

$$\times \leftarrow \lambda(x{:}R,\ y{:}R) \quad \mathrm{If} \left(\begin{array}{l}(x \geq 0 \wedge y \geq 0) \vee (x \leq 0 \wedge y \leq 0), \\ \mathrm{Abs}(x) \times_{\widetilde{\mathcal{R}}} \mathrm{Abs}(y), \\ \mathrm{Negative}(\mathrm{Abs}(x) \times_{\widetilde{\mathcal{R}}} \mathrm{Abs}(y))\end{array} \right)$$

Again $\tilde{\mathcal{R}}$ is a particular implementation of the real numbers and other implementations are possible. Again, to abstract away from the implementation we define the real number structure $\mathcal{R}$ as follows.

$$\mathcal{R} \equiv \mathrm{The}\left(G{:}\overline{\Gamma}\ \ G =_{\overline{\Gamma}} \widetilde{\mathcal{R}}\right)$$

$$\Gamma \equiv \overline{R{:}\mathbf{type}_1;\ +{:}R \times R \to R;\ \times{:}R \times R \to R}$$

7.7 The Complex Numbers

Chapter 8

Groups and Fields

There are two reasons for discussing groups and fields at this point. First, groups and fields are prominent examples of structure types. The preceding chapter on numbers constructed particular structures such as the integers and reals. While constructing particular structures is important, defining types of structures is also important and groups and fields provide a case study in types of structures. But the particular structure types of Groups and Fields also play a special role in mathematics. Every structure type $\overline{\Gamma}$ in mathematics associates each structure $G : \overline{\Gamma}$ with a group — the symmetry group of G. For $G : \overline{\Gamma}$ the $\overline{\Gamma}$ symmetry group of G is the set of $\overline{\Gamma}$-isomorphisms of G with itself. This is discussed in more detail below. Fields play a special role in defining "linearity", a concept central to much of mathematics.

8.1 Groups and Abelian Groups

We now define a group (in English) to be a set G together with a function $\bullet$ mapping two elements of G to an element of G where $\bullet$ satisfies the following conditions.

> **Associativity:** For all group elements a, b and c we have $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
>
> **Identity element:** There exists a group element e, called the identity element, such that for group element a we have $e \bullet a = a \bullet e = a$.
>
> **Inverse element:** For each group element a there exists a group element b, called the inverse of a, such that $a \bullet b = b \bullet a = e$.

We note that the identity element is unique — for any identity elements e and e' we have that $e = e \bullet e' = e'$. Hence the phrase "the identity element" is well-formed. We also have that inverses are unique — for any inverses b and b'

of a we have $b \bullet a = e = b' \bullet a$ and hence we have the following.

$$\begin{aligned}(b \bullet a) \bullet b &= (b' \bullet a) \bullet b \\ &= b' \bullet (a \bullet b) \\ &= b' \bullet e \\ &= b' \\ &= b \bullet (a \bullet b) \\ &= b\end{aligned}$$

Hence the phase "The inverse of a" is well formed. The inverse of a is generally written as a^{-1}.

A group is called *Abelian* if the group operation is also commutative — $x \bullet y = y \bullet x$. For an Abelian group it is traditional to write the group operation as $+$ rather than $\bullet$.

8.2 Permutation Groups and Representation

A permutation of a set S is a bijection from S to S. We can translate this into formal notation as

$$\mathbf{Permutation}[\sigma] \equiv \mathbf{Bijection}[\sigma, \sigma].$$

A permutation group is a subset G of the permutations of a set S where G is closed under functional composition and inverse. We can formalize this as

$$\mathbf{PermGroup} \equiv \overline{\alpha{:}\mathbf{type}_1; G{:}\mathbf{Permutation}[\alpha] \to \mathbf{bool};\ \mathcal{A}}$$

where $\mathcal{A}$ is a boolean expression stating that for $f, g{:}\mathbf{Permutation}[\alpha]$ if $f, g \in G$ (if $G(f)$ and $G(g)$) then $f \circ g \in G$ and if $f \in G$ then the inverse function f^{-1} is in G.

Although a permutation group is not literally (formally) a group, given a permutation group P we can define the group $G[P]$ as

$$G[P] \equiv \langle \alpha \leftarrow \mathbf{TheSubtype}(\mathbf{Permutation}[P.\alpha],\ P.G);\ \bullet \leftarrow \mathbf{The}(F{:}\alpha \times \alpha \to \alpha \mid \mathcal{A}) \rangle$$

where $\mathcal{A}$ states that F is function composition — that for $x : P.\alpha$ we have $F(f, g)(x) =_{P.\alpha} f(g(x))$. It is then possible to derive the following.

$$P{:}\mathbf{PermGroup} \vdash G[P]{:}\mathbf{Group}$$

A fundamental fact about groups is the following representation theorem which we will state in formal notation.

$$\vdash \forall G{:}\mathbf{Group}\ \exists P{:}\mathbf{PermGroup}\ \ G =_{\mathbf{Group}} G[P]$$

To prove this theorem er consider an arbitrary group G and consider the map from G to permutations of G defined by the following λ-expressions

$$B[G] \equiv \lambda x{:}G.\alpha \;\; \lambda y{:}G.\alpha \;\; x(G.\bullet)y$$

$B[G]$ is function from the elements of G to a corresponding permutations of the elements G. We can let $P[G]$ be the set of permutations which correspond to some element of G.

$$P[G] = \left\langle \begin{array}{l} \alpha \leftarrow G.\alpha; \\ G \leftarrow \{f{:}\mathbf{Permutation}[G.\alpha] \mid \exists x{:}G.\alpha \; f =_{\mathbf{Permutation[G}.\alpha]} B[G](x)\} \end{array} \right\rangle$$

We can then derive the following.

$$G{:}\mathbf{Group} \vdash P[G]{:}\mathbf{PermGroup}$$
$$\vdash \forall G{:}\mathbf{Group} \;\; G =_{\mathbf{Group}} G[P[G]]$$

8.3 Fields

Chapter 9

Vector Spaces

9.1 Basic Definition

Vector spaces and linear algebra play a central role in many areas of mathematics. In English one usually defines a vector space V over a field F to consist of a set of vectors plus two operations — vector addition mapping two vectors to a vector, and scalar multiplication mapping a scalar and a vector to a vector, and where these operations satisfy the axioms of a vector space given below. In formalizing this as a machine readable type expression we take the field F to be a free variables of the type and write the axioms as $\mathcal{A}$ (as usual).

$$\mathbf{VectorSpace}[F] \equiv \overline{\alpha:\mathbf{type}_1;\ +:\alpha\times\alpha\to\alpha;\ \bullet:\mathcal{F}.\alpha\times\alpha\to\alpha;\ \mathcal{A}}\ {}_{\mathcal{F}:\mathbf{Field}}(\langle\mathcal{F}\leftarrow F\rangle)$$

If V is a vector space over field F then the instances of $V.\alpha$ are called vectors and we generally write $x \in V$ or $x : V$ as abbreviations for $x : V.\alpha$. The field is typically either the real numbers or the complex numbers and the elements of the field will be called *scalars*. The symbol + will be used for both vector addition and scalar addition where we rely on context to distinguish which kind of addition is being used. We will typically use late alphabet letters such as x, y and z for vectors and early alphabet letters such as a, b and c for scalars. We will the scalar product of a scalar a and a vector x as ax. The vector space axioms are the following.

1. The vectors under vector addition form an Abelian group.
2. Associativity of scalar multiplication: $a(bx) = (ab)x$.
3. Distributivity over scalar sums: $(a + b)x = ax + bx$.
4. Distributivity over vector sums: $a(x + y) = ax + ay$
5. Scalar multiplicative identity: $1x = x$ where 1 is the multiplicative identity of F.

Since the vectors under vector addition form a group the vectors contain a unique additive identity element which we write as 0. Of course the field F also has an additive identity element 0. We rely on context to distinguish the scalar 0 from the vector 0. To show that $0x = 0$ it suffices to show that $y + 0x = y$. This can be done as follows where we write $-x$ for the inverse of x in the group of vectors under vector addition.

$$\begin{aligned} y + 0x &= (y + 0x) + (x + -x) \\ &= (y + -x) + (1x + 0x) \\ &= y + -x + (1 + 0)x \\ &= y + -x + 1x \\ &= y + -x + x \\ &= y \end{aligned}$$

Given that $0x = 0$ we can then show that $(-1)x = -x$. It suffices to show that $x + (-1)x = 0$. This can be done as follows.

$$x + (-1)x = 1x + (-1)x = (1 + (-1))x = 0x = 0$$

We will write $x - y$ for $x + (-y)$.

9.2 Basis of a Vector Space

A finite basis for a vector space V is a finite sequence $\langle e_1, \ldots, e_d \rangle$ where e_i is a vector in V such that we have the following properties.

- The basis vectors span the vector space — for any vector x there exist scalars $a_1, \ldots a_d$ such that $x = a_1 e_1 + \cdots a_d e_d$.
- No basis element can be written as a linear combination of other basis elements — for any scalar values $a_1, \ldots a_n$ such that some $a_i \neq 0$ we have that $a_1 e_1 + \cdots a_d e_d \neq 0$.

Note that if $a_i \neq 0$ but $a_1 e_1 + \cdots a_d e_d = 0$ then we have $e_i = -(1/a_i) \sum_{j \neq i} a_j e_j$. Hence the last condition is equivalent to the statement that no basis element can be written as a linear combination of other basis elements.

Theorem 2. *For any vector space V and basis $\langle e_1, \ldots e_d \rangle$ for V we have that any sequence $\langle u_1, \ldots, u_d \rangle$ of d linear independent vectors also spans V and hence forms a basis for V.*

Proof. Since $e_1, \ldots e_n$ spans V we must have $u_1 = a_1 e_1 + \cdots a_d e_d$. Since $u_1 \ldots$, u_d are linearly independent we must have $u_1 \neq 0$. Hence there is some $a_i \neq 0$ and we have

$$e_i = (1/a_i)(u_1 - \sum_{j \neq i} a_j e_j)$$

Hence any linear combination of the vectors e_1, ..., e_d can be converted to a linear combination of e_j for $j \neq i$ and u_1. So the vectors $\langle e_1, \ldots, e_{i-1}, u_1, e_{i+1}, \ldots, e_d \rangle$ span V. More generally, if $\langle e_1, \ldots, e_n, u_1, \ldots, u_m \rangle$ spans V and u_{m+1} is linearly independent of u_1, ..., u_m then we must have $u_{m+1} = a_1 e_1 + \cdots a_n e_n + b_1 u_1 + \cdots b_m u_m$ with some $a_i \neq 0$. This implies that

$$e_i = (1/a_i)(u_{m+1} - \sum_{j \neq i} a_j e_j - \sum_j b_j u_j)$$

But this implies that e_j for $j \neq i$ together with u_1, ..., u_{m+1} span V. We can continue to replace vectors e_i by vectors u_j until no vectors e_i remain. Hence the vectors u_1, ..., u_d span V. □

A vector space V will be said to be finite dimensional if there exists a finite basis for V. The above theorem implies that all bases of a finite dimensional vector space have the same length. Hence we can talk about the dimension of a finite dimensional vector space.

9.3 Basis-Relative Coordinates

Consider a basis $B = \langle e_1, \ldots, e_d \rangle$ for a d-dimensional vector space V. By the definition of a basis we have that e_1, ..., e_d span V and hence for any vector x we have that there exists a_1, ..., a_d such that $x = a_1 e_1 + \cdots a_d e_d$. We now note that the weights a_1, ... a_d are unique. To see this suppose that we have the following.

$$\begin{aligned} x &= a_1 e_1 + \cdots a_d e_d \\ x &= b_1 e_1 + \cdots b_d e_d \end{aligned}$$

By subtracting these two equations we get the following.

$$0 = (a_1 - b_1)e_1 + \cdots + (a_d - b_d)e_d$$

But by the linear independence property of a basis we must then have each weight in this combination is zero. Hence we have $a_i = b_i$ for all i.

Since the coefficients a_i are unique for a given basis B we can define $a_1^B(x)$, ..., $a_d^B(x)$ to be the unique scalars such that $x = a_1^B(x)e_1 + \cdots + a_d^B(x)e_d$. The scalars $\langle a_1^B(x), \ldots, a_d^B(x) \rangle$ will be called the coordinates of x relative to the basis B. We note that the coordinates of a sum of two vectors are obtained by adding the coordinate values. More specifically we have that if

$$\begin{aligned} x &= a_1 e_1 + \cdots + a_d e_d \\ y &= b_1 e_1 + \cdots + b_d e_d \end{aligned}$$

then

$$x + y = (a_1 + b_1)e_1 + \cdots + (a_d + b_d)e_d.$$

This can be written as follows.

$$a_i^B(x+y) = a_i^B(x) + a_i^B(y).$$

Similarly we have

$$a_i^B(cx) = ca_i^B(x)$$

It is very important to keep in mind that a different choice of basis vectors leads to a different "coordinate system" — the coordinates of a vector relative to one basis are different from the coordinates relative to a different basis. In a d-dimensional vector space, *any* linearly independent tuple of vectors e_1, ..., e_d forms a basis. In this basis the coordinates of e_i are $\langle 0, 0, \ldots, 0, 1, 0, \ldots, 0\rangle$ where the 1 occurs at position i. In particular, the coordinates of the vector e_1 in this basis are $\langle 1, 0, 0, \ldots, 0\rangle$. Any nonzero vector can be given as the first vector in some basis. Hence *any nonzero vector can be assigned the coordinates* $\langle 1, 0, 0, \ldots, 0\rangle$.

9.4 Voldemort in Vector Space

Here we will show that it is not possible to name a basis (or a coordinate system) for a vector space. To do this we must analyze the notion of isomorphism and symmetry for vector spaces.

Consider the following simple structure type which we will call a binary operation or **Binop**.

$$\mathbf{Binop} \equiv \overline{\alpha:\mathbf{type}_1;\ f:\alpha\times\alpha\to\alpha}$$

For two binary operations $F, G : \mathbf{Binop}$ we have $F =_{\mathbf{Binop}} G$ if there exists a bijection g from $F.\alpha$ to $G.\alpha$ which identifies $F.f$ with $G.f$. It is possible to show that the definition of identification given in chapter 5 corresponds to the requirement that the bijection g be such that for all $x, y : F.\alpha$ we have

$$g((F.f)(x,y)) = (G.f)(g(x), g(y)).$$

Now consider the type $\mathbf{VectorSpace}[F]$.

$$\mathbf{VectorSpace}[F] \equiv \overline{\alpha:\mathbf{type}_1;\ +:\alpha\times\alpha\to\alpha;\ \bullet:\mathcal{F}.\alpha\times\alpha\to\alpha;\ \mathcal{A}}\ {}_{\mathcal{F}:\mathbf{Field}}(\langle\mathcal{F}\leftarrow F\rangle)$$

For two vector spaces $V, W : \mathbf{VectorSpace}[F]$ we have $V =_{\mathbf{VectorSpace}[F]} W$ if there exists a bijection B from the vector type $V.\alpha$ to the vector type $W.\alpha$ that identifies $V.+$ with $W.+$ and $V.\bullet$ with $W.\bullet$. The condition that B identifies $V.+$ with $W.+$ is that for any $x, y : V.\alpha$ we have

$$g(x\ (V.+)\ y) = g(x)\ (W.+)\ g(y)$$

The condition that B identifies $V.\bullet$ with $W.\bullet$ is that for any $a : F.\alpha$and $x : V.\alpha$ we have

$$g(a(V.\bullet)x) = a(W.\bullet)\ g(x)$$

These two conditions can be written using context to distinguish the various sum and product operations as follows.

$$\begin{aligned} g(x+y) &= g(x)+g(y) \\ g(ax) &= ag(x) \end{aligned}$$

This is the normal definition of linear map g from a vector space V to a vector space W. We then get the following.

> Two vectors spaces over a field F are isomorphic if there exists a linear bijection between them. A symmetry of a vector space is linear bijection from that space to itself.

We first note that any two d-dimensional vector spaces over F are isomorphic. To see this let $B = \langle e_1, \ldots, e_d \rangle$ be a basis for a vector space V and let $B = \langle e'_1, \ldots, e'_d \rangle$ be a basis for a vector space W. We can then define a linear bijection from V to W by the following equation.

$$g(a_1 e_1 + \cdots + a_d e_d) = a_1 e'_1 + \cdots + a_d e'_d$$

In other words, for $x : V$ we define $g(x)$ to be the vector y in W such that the coordinates of x under basis B equal the coordinates of y under basis B'.

Now consider any two bases $B = \langle e_1, \ldots, e_n \rangle$ and $B' = \langle e'_1, \ldots, e'_d \rangle$ for a single vector space V. Define a mapping V from V to V by the following equation.

$$g(a_1 e_1 + \cdots + a_d e_d) = a_1 e'_1 + \cdots + a_d e'_d$$

The function g maps a point with coordinates $a_1, \ldots a_d$ under basis B to the point with those same coordinates in basis B'. Although the coordinates are the same, the points are different because the basis vectors (the coordinate systems) are different. This mapping is a symmetry of the vector space — it is a linear bijection from the space to itself. Furthermore, this symmetry maps the vector e_i to the vector e'_i. This symmetry "carries" the basis B to the basis B'. Hence the type **BasisFor**$[V]$ where V is a vector space has no canonical instance — all choices of a basis are equivalent. By Voldemort's theorem there is no way to name a basis (a coordinate system) for a vector space.

Chapter 10

Inner Products and the Gram-Schmidt Procedure

We now consider a vector space V over the reals. The inner product of two vectors is defined in high school classes in terms of the coordinates of those vectors. Given a basis (coordinate system) B We can define the inner product (or dot product) of two vectors x and y, which we will write as $xD_B\ y$, as follows.

$$xD_B\ y = a_1^B(x)a_1^B(y) + \cdots a_d^B(x)a_d^B(y)$$

This is just the normal coordinate-wise notion of inner product where $\langle a_1, \ldots, a_d\rangle \cdot \langle b_1, \ldots, b_d\rangle = a_1b_1 + \cdots a_db_d$. The inner product operation D_B is a function from two vectors to a scalar, we have $D_B : V \times V \to \mathcal{R}$. It is important to note however that different coordinate systems yield different inner product functions — if the basis vectors get half as large, the coordinates get twice as big. If the coordinates get twice as big the inner products get four times as big (for the same pairs of vectors). So for two different bases B and B' we have that D_B is different in general from $D_{B'}$.

We say that A is a bilinear function on V if $A : V \times V \to \mathcal{R}$ and A is linear in each of its two arguments. Namely we have the following.

$$\begin{aligned} A(x, ay + z) &= aA(x, y) + A(x, z) \\ A(ax + y, z) &= aA(x, z) + A(y, z) \end{aligned}$$

For a bilinear function A we will (for now) write $A(x, y)$ as yAx. In chapter 12 we will convert to a more standard "matrix notation" and write this as $y^\intercal Ax$. But the "transpose" of a vector is meaningless here and we will avoid that notation for now. In this notation the bilinearity of A can then be written as follows.

$$\begin{aligned} (ay + z)Ax &= a(yAx) + zAx \\ zA(ax + y)) &= azAx + yAx \end{aligned}$$

We now define a general notion of inner product. An inner product D on V is any symmetric, positive definite bilinear function on V. The symmetry property is that for any vectors x and y we have $xDy = yDx$ and the positive-definite property is that for any non-zero vector x we have $xDx > 0$. It is easy to check that for any basis B the inner product operation D_B satisfies these conditions. For any vector x and inner peoduct D we write $||x||_D$ (the norm of x under inner product D) for $\sqrt{xDx}$.

10.1 The Cauchy-Schwarz Inequality

Or first observation about inner products is the Cauchy-Schwartz inequality which states that for any inner product D on a vector space and any two vectors x and y we have

$$xDy \leq ||x||_D||y||_D.$$

We can prove the Cauchy-Schwartz inequality for an arbitrary inner product on an arbitrary vector space by first defining $\tilde{y}$ to be the unit vector $y/||y||_D$. We then define $x_{||}$ to be the vector $(xD\tilde{y})\tilde{y}$ and $x_\perp$ to be $x - x_{||}$. We then have $x = x_\perp + x_{||}$. Furthermore we have the following.

$$\begin{aligned} x_\perp D x_{||} &= (x - x_{||})Dx_{||} \\ &= xDx_{||} - x_{||}Dx_{||} \\ &= (xD\tilde{y})^2 - (xD\tilde{y})^2 \\ &= 0 \end{aligned}$$

We then have the following.

$$\begin{aligned} ||x||_D^2 &= (x_\perp + x_{||})D(x_\perp + x_{||}) \\ &= ||x_\perp||_D^2 + ||x_{||}||_D^2 \\ &\leq ||x_{||}||_D^2 \\ &= (xD\tilde{y})^2 \\ &= (xDy)^2/||y||_D^2 \end{aligned}$$

The last line is equivalent to the Cauchy-Schwarz inequality.

10.2 The Gram-Schmidt Procedure

Now consider a fixed but arbitrary inner product D on V. A basis $B = \langle e_1, \ldots, e_d \rangle$ will be called orthonormal under D if $e_i D e_i = 1$ for all i and $e_i D e_j = 0$ for $i \neq j$. We now observe that if B is an orthonormal basis under D the standard coordinate inner product D_B defined in terms of B-coordinates equals (as a function) the inner product D. To see this we can use bilinearity and orthonormality to prove the following for D.

$$(a_1e_1 + \cdots + a_de_d)D(b_1e_1 + \cdots + b_de_d) = a_1b_1 + \cdots a_db_d$$

Given an arbitrary inner product D and any basis $B = \langle e_1, \ldots, e_d \rangle$ one can use the Gram-Schmidt process to convert B to a basis $B' = \langle e'_1, \ldots, e'_d \rangle$ that is orthonormal for D. More specifically we define $e'_1, \ldots, e'_d$ using the following Gram-Schmidt procedure:

$$\begin{aligned} e'_1 &= e_1/||e_1||_D \\ e_{i+1,\perp} &= e_{i+1} - \sum_{j=1}^{i} e_{i+1} D e'_j \\ e'_{i+1} &= e_{i+1,\perp}/||e_{i+1,\perp}||_D \end{aligned}$$

By calculations similar to those used in proving the Cauchy-Schwartz inequality we can show that $e'_1, \ldots, e'_d$ is an orthonormal basis. This shows that for any inner product operation D there exists a basis B such that $D = D_B$. Every inner product operation is the standard coordinate-wise inner product in an appropriately chosen coordinate system.

10.3 Voldemort Challenges Inner Products

Of course for a given inner product D there exists a choice of d linear independent vectors which are not orthonormal under D. However, for any basis B, even those far from orthonormal under D, the basis B is always orthonormal under the inner product D_B defined in terms of B coordinates. Every basis is orthonormal in its own coordinate system.

Now consider any two inner product operations D and D' on V. By the above observations we have $D = D_B$ and $D' = D_{B'}$ for some coordinate systems B and B'. We showed in section 9.4 that for any two bases B and B' there exists a symmetry of the vector space V carrying B to B'. This same symmetry carries D to D'. Hence there is no canonical instance of the type **InnerProduct**$[V]$. By Voldemort's theorem, there is then no way to name any particular inner product operation for V.

Chapter 11

Gradients and the Dual Space

11.1 A Coordinate-Free Definition of Gradient and Dual Vector

Consider a vector space V over a field F and consider an arbitrary (possibly nonlinear) function $f : V \to F$ The function f is often called a scalar field — it assigns a scalar value to each point (each vector). A vector field assigns a vector value to each point. If the scalar field f is smooth then in very small neighborhoods of a vector x the function f is approximately linear and we have the following where $g : V \to F$ is linear.

$$f(x + \Delta x) \approx f(x) + g(\Delta x)$$

Here both x and Δx are vectors and the linear function g represents the gradient of the function f. There is no canonical way to write $g(\Delta x)$ as $(\nabla f) D \Delta x$ where ∇f is a gradient vector and D is an inner product operation because there is no canonical inner product (no inner product can be named). Instead, the gradient g is a vector in a different vector space — the dual space V^*. The dual space V^* is simply defined to be the set of linear functions $g : V \to F$.

We can now define the gradient of a smooth scalar field f as follows.

$$\left(\nabla_x f(x)\right)(\Delta x) = \lim_{\epsilon \to 0} \frac{f(x + \epsilon \Delta x) - f(x)}{\epsilon}$$

Note that if we double the vector Δx the value of the right hand side of the above expression doubles. The value should be linear in the vector Δx. We say that f is differentiable if the above limit always exists and there exists a g in V^* such that $\left(\nabla_x f(x)\right)(\Delta x) = g(\Delta x)$ for all Δx. If $f(x)$ is differentiable in x then $\nabla_x f(x)$ is a dual vector. This definition does not require coordinates and uses only the concepts available in a vector space.

The notation $\nabla_x\ f(x)$ is a special case of the more general notion $\nabla_x\ e[x]$ where $e[x]$ is an expression involving the variable x and possibly other variables as well. Hence the subscript in ∇_x is important — we have that $\nabla_x\ e[x,y]$ is different from $\nabla_y\ e[x,y]$. It is useful to consider the following more formal definition.

$$\nabla_x\ e[x,y] \equiv \lambda\ \Delta x{:}V\ \ \lim_{\epsilon\to 0} \frac{e[x+\epsilon\Delta x,y]-e[x,y]}{\epsilon}$$

We have that x remains a free variable of $\nabla_x\ e[x,y]$. For example, we will show in chapter 12 that $\nabla_x\ x^\intercal Ax$ equals $(A+A^\intercal)x$.

We now show that the dual space V^* itself forms a vector space with the same dimension as V. For $h,g:V^*$ we define $h+g$ to be the function defined by $(h+g)(x)=h(x)+g(x)$. One can check that if h is linear and g is linear then $h+g$ is linear. Similarly for $a:F$ and $g:V^*$ we define the function ag by $(ag)(x)=a(g(x))$. One can then show that the set of linear functions from V to F itself forms a vector space under this vector addition and scalar multiplication.

Recall that for a basis $B=\langle e_1,\ldots,e_d\rangle$ and a vector x we defined $a_i^B(x)$ to be the ith coordinate of x in the coordinate system defined by B. For the coordinate maps a_i^B we have $a_i^B(x+y)=a_i^B(x)+a_i^B(y)$ and $a_i^B(cx)=ca_i^B(x)$. Hence the mapping a_i^B from V to F is linear and we have $a_i^B:V^*$. It turns out that the coordinate maps $a_1^B,\ldots,a_d^B$ form a basis of V^* and hence the dual space V^* has the same dimension as V. To show this we must also show that these functions are linearly independent and span V^*. To show that they are linearly independent we simply note that for each coordinate function a_i^B we have that $a_i^B(e_i)=1$ while $a_j^B(e_i)=0$ for all $j\neq i$. Hence the function a_i^B cannot be expressed as a linear combination of a_j^B for $j\neq i$. To show that the coordinate functions span V^* we consider an arbitrary $g:V^*$. For an arbitrary g we have the following.

$$\begin{aligned} g(x) &= g(a_1^B(x)e_1+\cdots+a_d^B(x)e_d) \\ &= a_1^B(x)g(e_1)+\cdots a_d^B(x)g(e_d) \\ &= g(e_1)a_1^B(x)+\cdots g(e_d)a_d^B(x) \\ g &= g(e_1)a_1^B+\cdots+g(e_d)a_b^B \end{aligned}$$

We then have that in the coordinate system defined by B the coordinates of $g:V^*$ are $\langle g(e_1),\ldots,g(e_d)\rangle$ and the coordinates of x are $\langle a_1^B(x),\ldots,a_d^B(x)\rangle$ and $g(x)=g(e_1)a_1^B(x)+\cdots+g(e_d)a_d^B(x)$. This last equation states that for any basis B we have that $g(x)$ can be written as the "inner product" of the coordinates for g and the coordinates for x.

Note that if we double the length of every basis vector then the coordinates of the vectors get half as big while the coordinates of the dual vectors get twice as big but the "inner product" $g(x)$ retains its coordinate-free value. Vector coordinates are called contravariant coordinates (these coordinates get smaller when the basis vectors get bigger) and dual vector coordinates are called covariant coordinates (these coordinates get bigger when the basis vectors get bigger).

11.2 Voldemort Challenges Gradients

We now show that in dimension greater than one we cannot associate a dual vector in V^* with any particular direction in V. Note that if the vector space is one dimensional then there is only one direction V — we need at least two dimensions before no direction can be named. To simplify the discussion we consider finite dimensional vector spaces, but the claim holds in infinite dimension as well. For a finite dimensional vector space V of dimension greater than one and a dual vector $g : V^*$ define the type **AscentVector**$[V, g]$ to be subtype of the vectors x such that $g(x) = 1$. For any direction Δx we have that $(1/g(\Delta x))\Delta x$ is an ascent vector for g in the same direction as Δx. Now consider any dual vector g and any ascent vector Δx for g. One can show that there exists a basis $B_{\Delta x}$ for V in which $e_1 = x$ and $g(e_j) = 0$ for $j > 1$. In this basis the coordinates of g and the coordinates of x are both $\langle 1, 0, 0, \ldots, 0\rangle$. Now consider any two ascent vectors Δx and $\Delta x'$ for g and consider the two bases $B_{\Delta x}$ and $B_{\Delta x'}$. These two coordinates systems define a symmetry of V by mapping objects to their coordinates in $B_{\Delta x}$ and then mapping those coordinates back to objects using coordinate system $B_{\Delta x'}$. This symmetry maps g to itself — both coordinate systems assign the coordinates $\langle 1, 0, 0, \ldots, 0\rangle$ to g. But the symmetry carries Δx to $\Delta x'$. Hence all ascent vectors are symmetric images of each other and by Voldemort's theorem no ascent direction can be named. One can similarly show that, although V and V^* have the same dimension and are therefore isomorphic, there is no canonical isomorphism between V and V^* — if such an isomorphism could be named then we could name an ascent direction for a dual vector g.

We can also construct the dual of the dual $(V^*)^*$. Since this is the dual of V^* it must have the same dimension V^* and which equals the dimension of V. The vectors in $(V^*)^*$ are the linear functions from V^* to F. Note that for $x : V$ we can define the mapping $f_x : V^* \to F$ by $f_x(g) = g(x)$. It turns out that this mapping from V to $(V^*)^*$ is a linear bijection. Hence we *can* name a linear bijection between V and $(V^*)^*$ — there is a canonical (or natural) isomorphism between V and $(V^*)^*$.

Chapter 12

Coordinate-Free Linear Algebra

In chapter 10 we introduced the notation xDy for applying a bilinear function D to two vectors x and y. Bilinear functions are often represented by matrices — arrays of numbers. The expression xDy is reminiscent of matrix notation. We will develop linear algebra in parallel to matrix algebra but without the use of coordinate systems or arrays and where bilinear functions take the place of matrices.

12.1 Right Facing and Left Facing Vectors

In standard matrix algebra we typically have that a vector x is, by default, a "column vector" while $x^\intercal$ is a "row vector". This terminology is meaningless here has vectors are viewed as abstract points rather than as tuples of numbers. However, linear algebra requires marking vectors as "left facing" or "right facing". A left-facing vector combines with terms to the left and a right-facing vector combines with terms to the right. We will interpret a vector variable x without a transpose sign is left-facing while $x^\intercal$ as being right facing. If ω is a dual vector and x is a vector then $\omega^\intercal x$ and $x^\intercal \omega$ both denote the result of applying the function ω to the vector x. On the other hand $\omega x^\intercal$ is a bilinear function which takes a vector to the left and a dual vector to the right and we have $y^\intercal(\omega x^\intercal)\gamma = (y^\intercal\omega)(x^\intercal\gamma)$.

Dirac's Bra-ket notation is perhaps more intuitive than the transpose notation. In the bra-ket notation we write $|x>$ for x and $<x|$ for $x^\intercal$ and $x^\intercal Ay$ is written as $<x|A|y>$. Here $<x|$ is called a "bra" and $|y>$ is called "ket" and together they form a bracket. But the bra-ket notation also merely indicates which direction (right or left) a vector combines with other terms. Here we will continue to use the transpose notation because of its similarity to matrix notation.

12.2 Bilinear Functions Replace Matrices

Each of the arguments of a bilinear function can be either vectors or dual vectors. This gives the following four types of bilinear functions.

$$\begin{array}{lrcl} 1. & V \times V \to F & = & V \to V^* \\ 2. & V \times V^* \to F & = & V \to V \\ 3. & V^* \times V \to F & = & V^* \to V^* \\ 4. & V^* \times V^* \to F & = & V^* \to V \end{array}$$

Consider $A : V \times V \to F$. The type $V \times V \to F$ is the same as the type $V \to (V \to F)$ which (for bilinear functions) is the same as $V \to V^*$ as listed above. For $A : V \times V \to F$ and for $x : V$ we can let Ax denote the vector in V^* mapping y to $y^\intercal Ax$. This is consistent with thinking of A as a abstract matrix where Ax denotes a vector. Now consider $B : V \times V^* \to F$. This is the same as $V \to (V^* \to F)$ and V is naturally isomorphic to $V^* \to F$ we can think of this type as $V \to V$. Now consider $A : V \to V^*$ (type 1. above) and $B : V \to V$ (type 2. above). We can define AB to be the composition of the functions A and B which we can write as $(AB)x = A(Bx)$. We then have that AB has type $V \to V^*$ which is the same as $V \times V \to F$. This corresponds to matrix multiplication in matrix algebra. However, in coordinate-free linear algebra the product AB of two bilinear functions is only well defined when the output type of B (which is either V or V^*) matches the input type of A where the types are defined by the right hand column of the above table. This is equivalent to the statement that AB is defined when the right argument of A has type dual to the left argument of B.

12.3 Transpose and Inverse

If A is a bilinear function then we will write $A^\intercal$ for the bilinear function A but taking its arguments in the reverse order. The bilinear function $A^\intercal$ can be defined by the equation $x^\intercal(A^\intercal)y = y^\intercal Ax$. We have noted that a product AB of bilinear functions A and B is defined if the right argument of A is compatible with the left argument of B (the two argument types must be dual of each other). Note that if AB is defined then $B^\intercal A^\intercal$ is also defined and we have $(AB)^\intercal = B^\intercal A^\intercal$. Note that the transpose operation does not change the type of bilinear functions of types (1) and (4) above but switched types (2) and (3). So types (2) and (3) are equivalent — we can go from one to the other using the transpose operation.

We say that the bilinear function A is full rank if the map from x to Ax is a bijection between vector spaces where we view the type of A as being given by the second column in the above table (when we view A as a linear function between vector spaces). If A is full rank then we can define A^{-1} to be the inverse function and we then have that $A^{-1}A$ is the identity function on the vector space which is the right type of A. We also have that $AA^\intercal$ is the identity

function on the left type of A. Note that the inverse operation does not change the type of bilinear functions of type (2) or (3) but switches types (1) and (2).

> **Problem** 12.1: Show that if D is an inner product on V then $D : V \to V^*$ is full rank. More specifically, show that it is a bijection — that if $Dx = Dy$ then $x = y$.

We note that an inner product operation D is required to be positive definite and this implies that D is full rank — an inner product D on V defines a bijection from V to V^*. Hence, for any inner product D we have that D^{-1} is well defined and we have $D^{-1} : V^* \to V$ which is equivalent to $D^{-1} : V^* \times V^* \to F$. We then have that if D is a an inner product on V then D^{-1} is an inner product on V^*. So an inner product on V provides the same information as an inner product on V^*.

12.4 Computing a Gradient — Calculation Without Coordinates

We have now defined the operations underlying linear algebra. These operations satisfy associativity for products and distributivity of multiplacation over addition. For example, we have $(AB)C = A(BC)$ and $AB + AC = A(B + C)$. These algebraic manipulations allow calculation. As an example, we now calculate the gradient $\nabla_x \, x^\intercal Ax$.

$$\begin{aligned}
(\nabla_x \, x^\intercal Ax)(\Delta x) &= \Delta x^\intercal (\nabla_x \, x^\intercal Ax) \\
&= \lim_{\epsilon \to 0} \frac{(x + \epsilon \Delta x)^\intercal A(x + \epsilon \Delta x) - x^\intercal Ax}{\epsilon} \\
&= \lim_{\epsilon \to 0} \Delta x^\intercal Ax + x^\intercal A \Delta x + \epsilon \Delta x^\intercal A \Delta x \\
&= \Delta x^\intercal Ax + x^\intercal A \Delta x \\
&= \Delta x^\intercal Ax + \Delta x^\intercal A^\intercal x \\
&= \Delta x^\intercal (Ax + A^\intercal x) \\
\nabla_x \, x^\intercal Ax &= Ax + A^\intercal x
\end{aligned}$$

We can also (more easily) show the following which exhibits the significance of the subscript x in the notation ∇_x.

$$\nabla_x \; x^\intercal A y \;\; = \;\; Ay$$

$$\nabla_y \; x^\intercal A y \;\; = \;\; A^\intercal x$$

We can see that linear algebra, with variables over vectors, supports calculation in much the same way as ordinary algebra with variables over scalars.

Problem 12.2: Compute the gradient $\nabla_x f(x)$ where $f(x) = (x^T Ay)(z^\intercal Bx)$.

Problem 12.3: Compute $\nabla_x \sin(x^\intercal Ax)$.

Problem 12.4: Consider a vector space V over the reals and functions $f : \mathcal{R} \to \mathcal{R}$ and $g : V \to \mathcal{R}$. Use the definition of ∇_x to show that $\nabla_x f(g(x)) = \dot{f}(g(x)) \nabla_x g(x)$ where $\dot{f}$ of f is the first derivative of f. This is a form of the chain rule for differentiation.

12.5 Differentiating Vector Fields

Consider two vector spaces V and W and a function $F : V \to W$ so that for $x : V$ we have $F(x) : W$ but where $F(x)$ may be nonlinear in x. For example, if x is a point in physical space then $F(x)$ might be the gravitational field at x. We can define $\nabla_x F(x)$ using the same equation as before.

$$\nabla_x F(x) = \lambda \, \Delta x : V \;\; \lim_{\epsilon \to 0} \frac{F(x + \epsilon \Delta x) - F(x)}{\epsilon}$$

Here, however, $F(x)$ is a vector rather than a scalar. We then have that $\nabla_x F(x)$ is a linear mapping from vectors to vectors, or equivalently, a bilinear function on scalars (as in the table at the start of chapter 12). Again we have that in general x remains a free variable of $\nabla_x F(x)$.

In the case of where $f(x)$ is a scalar we have that $\nabla_x f(x)$ is a vector and vectors by default interact to the left so that we have

$$(\nabla_x f(x))(\Delta x) = \Delta x^\intercal (\nabla_x f(x)).$$

In the case where $F(x)$ is a vector we adopt the convention that $\nabla_x F(x)$ takes its argument on the right and we have

$$y^\intercal \left[(\nabla_x F(x))(\Delta x) \right] = y^\intercal \left[\nabla_x F(x) \right] \Delta x.$$

Problem 12.5: Consider three vector spaces V, W and H all over a field F. Consider (nonlinear) functions $F : W \to H$ and $G : V \to W$. Use the definition of ∇_x to derive the general chain rule $\nabla_x F(G(x)) = ((\nabla_y F(y))[y \leftarrow G(x)]) \, \nabla_x G(x)$.

12.6 Tensors

Chapter 13

Coordinate-Free Optimization

Consider a vector space V over the reals and a differentiable funcion $f : V \to \mathcal{R}$. If there exists a unique minimum of this function then we define $\text{argmin}_x\ f(x)$ to be the unique x^* such that for any vector $y \neq x^*$ we have $f(y) > f(x^*)$. It is also interesting to consider the case where the minimum is not unique, but for the sake of simplicity we will assume a unique minimizing point. We can use the gradient of f in a variety of ways in constructing algorithms for finding the minimum x^*. The optimization problem, and the gradient of f, are defined without reference to any coordinate system or inner product. We now consider some approaches to the optimization problem and consider the question of which approaches are defined purely in terms of the vector space structure and which require a (perhaps arbitrary) inner product.

13.1 Convexity

A (nonlinear) function $f : V \to R$ is called convex if for any two vectors $x, y : V$ and real number $\lambda \in [0, 1]$ we have the following.

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

Note that the definition of convexity does not assume any coordinate system or inner product. If f is convex then any local minimum of f is a global minimum of f.

13.2 The KKT Conditions

Given that f is differentiable, and that a unique minimizer x^* exists, we must have $\nabla_x f(x)$ is zero (the zero vector in V^*) at the point x^*. The simplest approach to finding x^* is to solve the equation $\nabla_x(f(x)) = 0$. In a given

coordinate system this can be written as d constraints in d unknown coordinates. However, the equation $\nabla_x(f(x)) = 0$ is coordinate free and can sometimes be solved in a coordinate free way using coordinate-free linear algebra. For example, if $f(x) = a + \omega^\intercal x + x^\intercal Ax$ then $x^* = (A + A^\intercal)^{-1}\omega$. No inner product or coordinate system is involved here.

The Karesh-Kuhn-Tucker (KKT) conditions handle the case of constrained optimization. Here we are interested in minimizing $f(x)$ subject to a system of constraints of the form $g_1(x) \leq 0,\ g_2(x) \leq 0, \ldots, g_k(x) \leq 0$ where each g_i is differentiable. We sat that a vector x satisfies the KKT conditions if there exists real numbers $\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_k$ with $\lambda_i \geq 0$ such that $\lambda_i = 0$ if $g_i(x) < 0$ (complementary slackness) and we have the following.

$$\nabla_x f(x) + \lambda_1 \nabla_x g_1(x) + \cdots + \lambda_k \nabla_x g_k(x) = 0$$

Note that every term in this sum is a dual vector. The KKT conditions are stated purely in terms of vector space structure and do not assume any coordinate system or inner product. One can show that any solution to the constrained optimization problem must satisfy the KKT conditions. If f and each g_i are convex then any solution to the KKT conditions is a global optimum.

13.3 Gradient Descent

Gradient descent is a widely used algorithm for finding the minimizer $x^* = \text{argmin}_x\ f(x)$. In gradient descent we compute a sequence of points $x_0,\ x_1,\ x_2,\ \ldots$ by "following the gradient" of f. Unfortunately, we showed in chapter 11 that the gradient vector $\nabla_x f(x)$, which is a dual vector, does not allow us to name any particular update direction for constructing x_{t+1} from x_t. The situation is different, however, if we are given an inner product D on V. Note that D^{-1} has type $V^* \times V^* \to \mathcal{R}$ which is the same as $V^* \to V$. Given an inner product D we can define the gradient descent algorithm by the following update equation where x_0 is selected in some unspecified way.

$$x_{t+1} = x_t - \eta_t D^{-1}(\nabla_x\ f(x))$$

Note that in a coordinate system defined by an orthonormal basis for D, we have that D has the coordinates of the identity matrix. But the above update equation is meaningful independent of any choice of any coordinates. Here $\eta_t > 0$ is a learning rate and one typically requires that $\sum_t \eta_t = \infty$ but $\sum_t \eta_t^2$ is finite in which case we have that if f is strictly convex then $\lim_{t\to\infty} x_t = x^*$. This convergence of x_t to x^* is independent of the choice of D. However, the rate of convergence of x_t to x^* depends on the choice of D.

13.4 Newton's Method

Although there is no canonical inner product for a vector space V, given a strictly convex scalar field f we can name an appropriate inner product to use

in gradient descent on f. In particular, if f is strictly convex then the Hessian of f — the second derivative of f — defines an inner product on V. We now define the second derivative of f.

Note that if $f(x)$ is a scalar field then $\nabla_x f(x)$ is a vector field. For example, we have that $x^\intercal Ax$ is a nonlinear scalar field. In this case we have

$$\nabla_x \ x^\intercal Ax = (A + A^\intercal)x$$

and hence $\nabla_x \ x^\intercal Ax$ is a vector field. Since, in general, $\nabla_x f(x)$ is a vector field, we can differentiate it again and write $\nabla_x \nabla_x f(x)$. We can use the definition of the gradient of a vector field to derive the following.

$$\begin{aligned} \nabla_x \nabla_x \ x^\intercal Ax &= \nabla_x \left[(A + A^\intercal)x\right] \\ &= A + A^\intercal \end{aligned}$$

The second derivative $\nabla_x \nabla_x f(x)$ is called the Hessian of f and we will abbreviate this as $H_f[x]$ (note that in general x remains a free variable of $\nabla_x \nabla_x f(x)$). We can write the second order Taylor expansion of a scalar field $f(x)$ as follows.

$$f(x + \Delta x) \approx f(x) + (\nabla_x f(x))^\intercal \Delta x + \frac{1}{2} \Delta x^\intercal H_f[x] \Delta x$$

Note that the second order Taylor expansion is defined independent of any choice of coordinates or choice of inner product. For a scalar field $f(x)$ we have that the Hessian $H_f = \nabla_x \nabla_x f(x)$ has type $V \times V \to \mathcal{R}$. It can also be shown that H is symmetric. A doubly differentiable scalar field $f(x)$ is called strictly convex if H_f is positive definite. Note that we have that H_f is positive definite if and only if for all $\Delta x \neq 0$ the second order term in the second order Taylor expansion of f is positive. We now have that if f is strictly convex then H_f is an inner product on V.

Now again consider the problem of minimizing a scalar field $f(x)$. Let $G_f[x]$ abbreviate $\nabla_x f(x)$ — $G_f[x]$ is the gradient of f at the point x. We can write the first order Taylor expansion of $G_f[x]$ as follows.

$$G_f[x + \Delta x] \approx G_f[x] + H_f[x] \Delta x$$

At the minimum x^* we must have $G_f[x^*] = 0$. The above approximate expression for $G_f(x + \Delta x)$ can be solved for x^* as follows.

$$\begin{aligned} G_f[x] + H_f[x] \Delta x &= 0 \\ \Delta x &= -H_f[x]^{-1} G_f[x] \\ x^* &\approx x - H_f[x]^{-1} G_f[x] \end{aligned}$$

Newton's method algorithm initializes x_0 in some unspecified way and then repeats the following update.

$$x_{t+1} = x_t - H_f[x_t]^{-1} G_f[x_t]$$

When x_0 is far from x^* this update is not guaranteed to converge. We can ensure convergence by converting this to a form of gradient descent with an appropriate learning rate η_t.

$$x_{t+1} = x_t - \eta_t H_f[x_t]^{-1} G_f[x_t]$$

This is analogous inner-product gradient descent but where the inner product used in computing x_{t+1} is taken to be $H_f(x_t)$. This is sometimes referred to as second order gradient descent. Methods exploiting H_f are generally called second order methods. Second order methods generally require fewer iterations, although computing H_f with respect to a particular coordinate system can be expensive in high dimension.

> **Problem** 13.1: Let V be a vector space and consider a nonlinear function $f : V \to R$ of the form $f(x) = a + u^T x + x^T M x$ with $a : R$, $u : V^*$ and M a full-rank bilinear function $M : V \times V \to R$. We consider applying Newton's method to minimize f starting at the point x_0. Give the Newton method update for computing x_{i+1} from x_i in terms of a, u and M. (Note that M need not be symmetric.) How quickly will Newton's method converge in this case?

13.5 Convex Duality

Chapter 14

Coordinate-Free Least Squares Regression

14.1 Classical Regression

We now consider a vector space V over the reals and a sequence of pairs $\langle x_1, y_1\rangle$, ..., $\langle x_N, y_N\rangle$ with $x_i : V$ and y_i a real number. We are interested in fitting a linear function to these points. In particular we want to find a "weight vector" $\omega : V^*$ such that $\omega^\intercal x_i$ is near y_i. Some formulation of regression include a bias term so that the predictor of y_i has the form $\omega^\intercal x_i + b$. However, it is always possible to include a constant feature in x_i so that the bias term b is simply one of the coordinates of *omega*. We then have that predictors of the form $\omega^\intercal x_i$ suffice. In this case the problem we are interested in solving can be written as follows.

$$\omega^* = \operatorname*{argmin}_{\omega} \sum_{i=1}^{N} (\omega^\intercal x_i - y_i)^2$$

We will assume here that the minimum is unique and that the argmin is well defined. This is the classical least squares regression problem. Note that ω^* is defined without any reference to a coordinate system or an inner product operator. So it should be possible give an expression for ω^* which also does not involve any coordinate system or inner product.

To solve this problem we note that $\sum_{i=1}^{N} (\omega^\intercal x_i - y_i)^2$ is scalar field as a function of the vector ω. We can find the minimum by setting the gradient to zero. Using the definition of gradient one can prove that for $g : \mathcal{R} \to \mathcal{R}$ and $f : V \to R$ we have the following where $g'(z)$ is the derivative of g at the scalar value z.

$$\nabla_x g(f(x)) = g'(f(x)) \nabla_x f(x)$$

This is a generalization of the chain rule of differentiation. Applying this rule to $(\omega^\intercal x_i - y_i)^2$ gives the following.

$$\nabla_\omega (\omega^\intercal x_i - y_i)^2 = 2(\omega^\intercal x_i - y_i) x_i$$

We can now solve for ω^* as follows.

$$\begin{aligned} 0 &= \nabla_\omega \sum_i (\omega^\intercal x_i - y_i)^2 \\ &= \sum_i 2(\omega^\intercal x_i - y_i) x_i \\ 0 &= \sum_i x_i (x_i^\intercal \omega - y_i) \\ \left(\sum_i x_i x_i^\intercal\right) \omega &= \sum_i y_i x_i \\ \omega^* &= \left(\sum_i x_i x_i^\intercal\right)^{-1} \left(\sum_i y_i x_i\right) \end{aligned}$$

This expression for ω^* indeed does not make use of any coordinate system or inner product operation.

14.2 Ridge Regression

If the data points $\langle x_1, y_1\rangle$, ..., $\langle x_N, y_n\rangle$ are such that x_1, ..., x_N are linearly independent then we can select a basis in which each x_i is a basis vector. In this case there trivially exists $\omega : V^*$ such that $\omega^\intercal x_i = y_i$ for all data points $\langle x_i, y_i\rangle$. But in this case there is no reason to believe that least square regression will produce a predictor that is of value on a new input point x_i. In this case it is better to impose a prior bias on the weight vectors ω and prefer those weight vectors ω that are a-priori favored. This can be done by introducing a inner product operation D on V and consider the following optimization problem.

$$\omega^* = \operatorname*{argmin}_\omega \sum_i \frac{1}{2}(\omega^\intercal x_i - y_i)^2 + \frac{1}{2}\lambda \omega^\intercal D^{-1} \omega$$

Here we prefer vectors $\omega : V^*$ with small size as determined by the inner production D. This is called ridge-regression and the choice of the inner product D corresponds to the choice of an a-prior bias on weight vectors. We can solve the ridge-regression optimization problem as follows.

$$\begin{aligned}
0 &= \nabla_\omega \sum_i \frac{1}{2}(\omega^\intercal x_i - y_i)^2 + \frac{1}{2}\lambda\omega^\intercal D^{-1}\omega \\
&= \sum_i (\omega^\intercal x_i - y_i)x_i + \lambda D^{-1}\omega \\
&= \sum_i x_i(x_i^\intercal \omega - y_i) + \lambda D^{-1}\omega \\
\left(\sum_i y_i x_i\right) &= \left(\sum_i x_i x_i^\intercal + \lambda D^{-1}\right)\omega \\
\omega &= \left(\sum_i x_i x_i^\intercal + \lambda D^{-1}\right)^{-1}\left(\sum_i y_i x_i\right)
\end{aligned}$$

In summary, while least squares regression is defined independent of any choice of inner product on the vector space, ridge regression requires a choice of inner product to define an a-priori bias on the weight vectors ω.

14.3 Voldemort Challenges Regression

Problem 14.1: **Voldemort's no free lunch theorem for linear regression.** Let V be a d-dimensional vector space and let $\langle x_1, y_1\rangle$, ..., $\langle x_n, y_n\rangle$ be pairs with $x_i \in V$ and $y_i \in \mathcal{R}$. Prove that for $n < d$ it is not possible to name any nonzero linear function from V to $\mathcal{R}$ (any nonzero dual vector) in terms of the structure of V and the given pairs.

Chapter 15

A Coordinate-Free Central Limit Theorem

To state the central limit theorem on a vector space we will need to be able to express integrals on a vector space and probability distributions on a vector space. A rigorous treatment of these concepts in beyond the scope of this book. However, it worth noting that the central limit theorem can be stated for a vector space without assuming any given inner product operation. Here we present this fact in an very informal and intuitive way.

Consider a probability distribution (formally a finite measure) π on a d-dimensional vector space V over the reals. For a function $f(x)$ of a vector space x we will write $E_{x\sim\pi}[f(x)]$ for the expectation of $f(x)$ when drawing x according to the distribution π. Assuming that π has a first moment there exists an expected (or mean) vector $\mu = E_{x\sim\pi}[x]$. We can estimate μ by computing an empirical average $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ where each x_i is drawn independently from distribution π. Because each x_i is a random variable, the mean estimate $\hat{\mu}$ is also a random variable. The central limit theorem says that as n goes to infinity the distribution of $\hat{\mu}$ approaches that of a multivariate Gaussian distribution.

Note that the distribution of the mean estimate $\hat{\mu}$ is determined by the distribution π,the vector space V, and the sample size n. It must therefore be independent of any arbitrary choice of coordinate system or an inner product for V. The challenge here is to state the central limit theorem without introducing arbitrary coordinates or an arbitrary inner product operation. Our first observation is that if the distribution π has finite second moment we can define the a bilinear function analogous to the covariance matrix as follows.

$$\Sigma = E_{x\sim\pi}\left[(x-\mu)(x-\mu)^\intercal\right]$$

We then have that Σ has type $V^* \times V^* \to \mathcal{R}$ and we have the following for dual vectors α and β.

$$\alpha^\intercal \Sigma \beta = E[(\alpha^\intercal x - E[\alpha^\intercal x])(\beta^\intercal x - E[\beta^\intercal x])]$$

This equation says that $\alpha^t \Sigma \beta$ equals the covariance of the random variables $\alpha^\intercal x$ and the random variable $\beta^\intercal x$ (hence the term covariance matrix). In particular we have that $\alpha^\intercal \Sigma \alpha$ is the variance of the random variable $\alpha^\intercal x$. Note that no coordinates are involved in these concepts.

It is easy to see that the bilinear function Σ is symmetric and positive semidefinite meaning that $\alpha^\intercal \Sigma \alpha \geq 0$. We will now assume that Σ is positive definite which will be true unless the distribution π is restricted to a subspace of V of dimension less than d. When Σ is positive definite we have that Σ^{-1} is an inner product on V.

We now argue informally that an inner product D on V naturally assigns volume (measure) to regions (subsets) of V. A measure ν assigns a weight $\nu(S)$ to each measurable subset S of V. A measure ν on V will be called uniform if for any (measurable) subset S of V and vector x we have $\nu(S + x) = \nu(S)$ where $S + x$ is the set of vectors of the form $y + x$ for $y \in S$. It can be shown that any two uniform measures ν and ν' on V are related by a constant — there exists c such that for all S we have $\nu'(S) = c\nu(S)$. Now consider a basis $B = \langle e_1, \ldots, e_d \rangle$ which is orthonormal for D. Define the unit box for this basis to be the set of vectors x where each coordinate relative to this basis is in $(0, 1)$ — the set of vectors x with $0 \leq a_i^B(x) \leq 1$. We can define ν_D to be the unique uniform measure on V such that a unit D-box has measure 1. The measure ν_D depends only on the inner product D and not on the choice of basis used to define the D-box.

If ν is a measure assigning volume to subsets of V, and $f(x)$ is a function from V to the reals, and S is a region of (subset of) V then it is possible to define the integral $\int_S f(x)\nu(dx)$ which is the integral over the region S of the function $f(x)$. This integral can be defined as the limit, as we partition V into finer and finer grids, of the sum over all the grid cells of the size of the cell times $f(x)$ where x is a vector in the cell (the Riemann integral). Here the notation $\nu(dx)$ is meant to express the size of very small grid cell dx under the measure ν. A formal definition of integrals is beyond the scope of this book.

We can now express the central limit theorem as follows. For any distribution π on V with finite second moments, and for any measurable subset S of V, we have the following.

$$\lim_{n\to\infty} P_{x_1,\ldots,x_n\sim\pi}\left[\sqrt{n}(\hat{\mu} - \mu) \in S\right] = \int_S (2\pi)^{-d/2} e^{-\frac{1}{2}(\hat{\mu}-\mu)^\intercal \Sigma^{-1}(\hat{\mu}-\mu)} \nu_{\Sigma^{-1}}(dx)$$

This notation is now free of any choice of coordinates or choice of ambient inner product.

In practice calculations are usually made relative to some given (perhaps arbitrary) coordinate system imposing some (perhaps arbitrary) inner product D. We can perform calculations in the ambient coordinate system by noting the following relation ship between two measures ν_D and $\nu_{D'}$.

$$\nu_{D'}(dx) = \sqrt{|D^{-1}D'|}\, \nu_D(dx)$$

Note that $D^{-1}D'$ has type $V \to V$ and we can define the "determinant" $|D^{-1}D'|$

to be the product of the eigenvalues of this mapping. Eigenvectors and eigenvalues are discussed the chapter 16. In particular we have the following.

$$\nu_{\Sigma^{-1}}(dx) = \sqrt{|D^{-1}\Sigma^{-1}|}\,\nu_D(dx)$$

If D is the inner product of the ambient coordinate system then the coordinates of D as a matrix in the ambient coordinate system are those of identity matrix and the central limit theorem takes on the traditional form involving the leading factor of $1/\sqrt{|\Sigma|}$ in the leading constant of the Gaussian distribution.

Chapter 16

Principle Component Analysis (PCA)

16.1 Problem Statement

In PCA we are interested in dimensionality reduction. We want to map vectors in a d-dimensional space into vectors in a k-dimensional vector space with $k < d$ and often where k is much smaller than d. Dimensionality reduction can be used in wide variety of applications where it is more convenient to work with lower dimensional vectors. One example is the ridge regression algorithm described in chapter 14. Ridge regression, and regularization in general, is important when the dimension of the space is comparable to, or larger than, the sample size. In such cases an alternative to regularization is to reduce the dimension of the space and then perform unregularized (normal) regression.

To define PCA we assume a set of vectors $x_1, \ldots, x_N$. We will assume that these are drawn IID from some probability distribution on vectors. We define the following.

$$\begin{aligned} \mu &= \mathrm{E}[x] \\ \hat{\mu} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ \Sigma &= \mathrm{E}[(x-\mu)(x-\mu)^\intercal] \\ \hat{\Sigma} &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\hat{\mu})(x_i-\hat{\mu})^\intercal \end{aligned}$$

In PCA we often have $n << d$ which implies that the bilinear function $\hat{\Sigma}$ is not full rank and hence does not have an inverse. This can be contrasted with the

central limit theorem in chapter 15 where we hold d fixed and take the limit as $n \to \infty$. The reason for using $1/(n-1)$ rather than $1/n$ in the definition of $\hat{\Sigma}$ is given by the following calculation (note that we are not using coordinates).

$$\begin{aligned}
\mathrm{E}\left[\hat{\Sigma}\right] &= \frac{1}{n-1}\mathrm{E}\left[\sum_i (x_i - \hat{\mu})(x_i - \hat{\mu})^\intercal\right] \\
&= \frac{1}{n-1}\mathrm{E}\left[\sum_i ((x_i - \mu) - (\hat{\mu} - \mu))((x_i - \mu) - (\hat{\mu} - \mu))^\intercal\right] \\
&= \frac{1}{n-1}\begin{pmatrix} \mathrm{E}\left[\sum_i (x_i - \mu)(x_i - \mu)^\intercal\right] \\ -\mathrm{E}\left[\sum_i (\hat{\mu} - \mu)(x_i - \mu)^\intercal\right] \\ -\mathrm{E}\left[\sum_i (x_i - \mu)(\hat{\mu} - \mu)^\intercal\right] \\ +\mathrm{E}\left[(\hat{\mu} - \mu)(\hat{\mu} - \mu)^\intercal\right] \end{pmatrix} \\
&= \frac{1}{n-1}\left(n\Sigma - \Sigma - \Sigma + \Sigma\right) \\
&= \Sigma
\end{aligned}$$

A linear subspace W of V is defined to be a subset of V closed under scaling and vector addition. Note that any linear subspace of V must contain the zero vector. A linear subspace W can be visualized as a kind of hyperplane in V passing through the origin (the zero vector). Given the inner product D on V we can define the projection of a point x into a linear subspace W, which we will denote as $x \mapsto_D W$, as follows.

$$x \mapsto_D W = \operatorname*{argmin}_{y \in W} ||x - y||_D$$

16.2 Problem Solution

The objective in PCA is to find a k-dimensional linear subspace W minimizing the sum of the squared distance of from $x_i - \hat{\mu}$ to the nearest point in W. To simplify the notation we now write Δx_i for $x_i - \hat{\mu}$ and define W^* as follows.

$$W^* = \operatorname*{argmin}_W \sum_i || \Delta x_i - (\Delta x_i \mapsto_D W) ||_D^2$$

For orthonormal vectors $e_1, \ldots e_k$ (relative to D) let $W(e_1, \ldots, e_k)$ be the linear subspace spanned by $e_1, \ldots, e_k$. One can then show the following.

$$\Delta x \mapsto_D W(e_1, \ldots, e_k) = (\Delta x^\intercal D e_1)e_1 + \cdots + (\Delta x^\intercal D e_k)e_k$$

The optimization problem defining W^* can then be recast as follows where e_1,

$\ldots$ e_k are constrained to be orthonormal.

$$\begin{aligned} e_1^*, \ldots e_k^* &= \underset{e_1,\ldots,e_k}{\operatorname{argmin}} \sum_i ||\Delta x_i - \Delta x_i \mapsto_D W(e_1, \ldots, e_k)||_D^2 \\ &= \underset{e_1,\ldots,e_k}{\operatorname{argmin}} \sum_i ||\Delta x_i||_D^2 - ||\Delta x_i \mapsto_D W(e_1, \ldots, e_k)||_D^2 \\ &= \underset{e_1,\ldots,e_k}{\operatorname{argmax}} \sum_i ||\Delta x_i \mapsto_D W(e_1, \ldots, e_k)||_D^2 \\ &= \underset{e_1,\ldots,e_k}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{j=1}^{k} (\Delta x_i^\intercal D e_j)^2 \end{aligned}$$

In other words, we want to maximize the sum of the norm squared of the projections of the points Δx_i into the subspace $W(e_1, \ldots, e_k)$. We can rewrite this maximization problem as follows.

$$\begin{aligned} e_1^*, \ldots e_k^* &= \underset{e_1,\ldots,e_k}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{j=1}^{k} (\Delta x_i^\intercal D e_j)^2 \\ &= \underset{e_1,\ldots,e_k}{\operatorname{argmax}} \sum_{j=1}^{k} \sum_{i=1}^{n} (e_j^\intercal D \Delta x_i)(\Delta x_i^\intercal D e_j) \\ &= \underset{e_1,\ldots,e_k}{\operatorname{argmax}} \sum_{j=1}^{k} e_j^\intercal D \left(\sum_{i=1}^{n} \Delta x_i \Delta x_i^\intercal \right) D e_j \\ &= \underset{e_1,\ldots,e_k}{\operatorname{argmax}} \sum_{j=1}^{k} e_j^\intercal D \hat{\Sigma} D e_j \end{aligned}$$

We now consider the case of $k = 1$ — the case of reduction to a single dimension. In this case we can solve for e_1 using the KKT conditions where we set the gradient of the object $e_1^\intercal D \hat{\Sigma} D e_1$ to λ times the gradient of the constraint $||e_1||_D^2 = 1$ giving the following.

$$D \hat{\Sigma} D e_1 = \lambda D e_1$$

The inner product D is full rank and hence we can multiply both sides by D^{-1} yielding the following.

$$\hat{\Sigma} D e_1 = \lambda e_1$$

Note that $\hat{\Sigma} D$ has type $V \to V$. The above condition states that e_1 is an eigenvector of this map with eigenvalue λ. In a coordinate system where D has the coordinates of the identity matrix we are requiring that the coordinates of e_1 form an eigenvector of the covariance matrix $\hat{\Sigma}$.

We now show that since $\hat{\Sigma}$ is symmetric and positive semidefinite we have that eigenvectors of $\hat{\Sigma} D$ with distinct eigenvalues are orthogonal under D. This

can be shown as follows where e_1 and e_2 are eigenvectors of $\hat{\Sigma}D$ with eigenvalues λ_1 and λ_2 respectively.

$$\begin{aligned} e_1^\intercal D\hat{\Sigma}De_2 &= e_1^\intercal D(\lambda_2 e_2) \\ &= \lambda_2 e_1^\intercal De_2 \\ &= e_2^\intercal D\hat{\Sigma}De_1 \\ &= e_2^\intercal D(\lambda_1 e_1) \\ &= \lambda_1 e_1^\intercal De_2 \end{aligned}$$

Now if $\lambda_1 \neq \lambda_2$ then we must have $e_1^\intercal De_2 = 0$. It can be shown that the solution to the PCA optimization problem is to take $e_1, \ldots, e_k$ to be the k eigenvectors of $\hat{\Sigma}D$ with the largest eigenvalues.

Problem 16.1: Let V be a vector space and let $D : V \times V \to R$ be a symmetric and positive definite abstract matrix. Let $\Sigma : V^* \times V^* \to R$ also be symmetric and positive definite. Typically D is an inner product and Σ is a covariance matrix. Positive definite symmetric operators (as opposed to positive seimidefinite operators) are always full rank and hence always invertable (you do not have to prove this). Let $u : V^*$ and $v : V^*$ be eigenvectors of $D\Sigma$ with distinct eigenvalues. In other words we have the following where $\lambda_1 \neq \lambda_2$.

$$\begin{aligned} D\Sigma u &= \lambda_1 u \\ D\Sigma v &= \lambda_2 v \end{aligned}$$

Show that u and v are orthogonal as measured by D^{-1}, i.e., show $u^T D^{-1} v = 0$.

Problem 16.2: Let D_1 and D_2 be two different inner product operations on the same vector space V. Define an abstract matrix of type $V \to V$ in terms of D_1 and D_2 and describe how this might be used for dimentionality reduction.

16.3 Voldemort Challenges Dimensionality Reduction

Problem 16.3: **Voldemort challenges inner products again.** Let V be a d-dimensional vector space and let $x_1, \ldots, x_n$ be vectors in V. Prove that for $n < d$ it is not possible to name any inner product D on V in terms of the vectors $x_1, \ldots, x_n$. More specifically, show that $n < d$ and for any inner product D on V there exists an distinct inner product D' which behaves identically to D on the points $x_1, \ldots, x_n$. Explain why Voldemort's theorem then implies that no such inner product can be named.

Problem 16.4: **Voldemort's no free lunch theorem for dimensionality reduction.** Let V be a d-dimensional vector space and let $x_1, \ldots, x_n$ be vectors in V. Prove that for $n < d$ it is not possible to name any nonzero linear map from V to the space $W(x_1, \ldots, x_n)$ of vectors spanned by $x_1, \ldots, x_n$.

Chapter 17

Canonical Correlation Analysis (CCA)

Canonical correlation analysis is a dimensionality reduction method similar to principal component analysis (PCA) but which does not involve a notion of distortion defined independent of the probability distribution on vectors. Rather than assume a probability distribution on a vector space we assume a distribution on pairs (x, y) where x and y are vectors in different vector spaces. We want to find a low dimensional representational of x which carries as much information about y as possible.

17.1 Problem Statement

For any two real-valued random variables x and y we define the correlation $\rho_{x,y}$ as follows.

$$\begin{aligned} \rho_{x,y} &= \frac{\mathrm{E}\left[(x-\overline{x})(y-\overline{y})\right]}{\sigma_x \sigma_y} \\ \\ \overline{x} &= \mathrm{E}\left[x\right] \\ \overline{y} &= \mathrm{E}\left[y\right] \\ \\ \sigma_x &= \sqrt{\mathrm{E}\left[(x-\overline{x})^2\right]} \\ \sigma_y &= \sqrt{\mathrm{E}\left[(y-\overline{y})^2\right]} \end{aligned}$$

Note that $\rho_{x,y}$ is invariant to scaling x — if we define $x' = \alpha x$ for $\alpha > 0$ then we get $\mathrm{E}\left[x'y\right] = \alpha \mathrm{E}\left[xy\right]$ and $\sigma_{x'} = \alpha\sigma_x$ and hence $\rho_{x',y} = \rho_{x,y}$. By a form of the Cauchy-Schwarz inequality we have $|\mathrm{E}\left[(x-\overline{x})(y-\overline{y})\right]| \leq \sigma_x\sigma_y$ and we have $\rho_{x,y} \in [-1, 1]$.

Now let V and W be vector spaces and let $\mathcal{P}$ be a distribution on $V \times W$. In other words, $\mathcal{P}$ is a probability distribution on pairs (x, y) with $x \in V$ and $y \in W$. For fixed vectors $u \in V^*$ and $v \in W^*$ we have that $u^T x$ and $v^T y$ are real valued random variables and we can consider their correlation $\rho_{(u^T x),(v^T y)}$. It is possible to show that this correlation can be expressed as follows.

$$\begin{aligned}
\rho_{u,v} &= \frac{\mathrm{E}\left[u^T(x-\overline{x})(y-\overline{y})^T v\right]}{\sqrt{\mathrm{E}\left[u^T(x-\overline{x})(x-\overline{x})^T u\right]\mathrm{E}\left[v^T(y-\overline{y})(y-\overline{y})^T v\right]}} \\
&= \frac{u^T \Sigma_{VW} v}{\sqrt{u^T \Sigma_{VV} u \; v^T \Sigma_{WW} v}} \\
\Sigma_{VW} &= \mathrm{E}\left[(x-\overline{x})(y-\overline{y})^T\right] \\
\Sigma_{VV} &= \mathrm{E}\left[(x-\overline{x})(x-\overline{x})^T\right] \\
\Sigma_{WW} &= \mathrm{E}\left[(y-\overline{y})(y-\overline{y})^T\right]
\end{aligned}$$

Canonical correlation analysis computes a sequence of pairs (u_1, v_1), ..., (u_k, v_k) with $u_i \in V^*$ and $v_i \in W^*$ are the solution to the following optimization problem.

Maximize ρ_{u_i,v_i} subject to:

$$u_i \in V^*, \;\; v_i \in W^*$$

$$u_i^T \Sigma_{VV} u_i = 1, \;\; v_i^T \Sigma_{WW} v_i = 1$$

$$u_i^T \Sigma_{VV} u_j = 0, \;\; v_i^T \Sigma_{WW} v_j = 0 \;\text{ for all } j < i$$

17.2 Problem Solution

We can then represent $x \in V$ by the vector $(u_1^T x, \; u_2^T x, \; \ldots, \; u_k^T x) \in R^k$. This reduces dimension when k is smaller than the dimension of V.

We now consider the optimization problem defining (u_1, v_1). In this case we want to simply maximize $\rho_{u,v}$. We first consider the problem of optimizing v given u which is the following optimization problem.

For $u \in V^*$ fixed, maximize $u^T \Sigma_{VW} v$ for $v \in W^*$ such that $v^T \Sigma_{WW} v = 1$.

Setting the gradient of the objective function equal to λ times the gradient of the constraint gives the following.

$$\begin{aligned}
\Sigma_{WV} u &= 2\lambda \Sigma_{WW} v \\
v &= \frac{1}{2\lambda} \Sigma_{WW}^{-1} \Sigma_{WV} u
\end{aligned}$$

Applying the constraint that $v^T \Sigma_{WW} v = 1$ gives the following optimal value $v^*(u)$ as a function of u.

$$\begin{aligned} v^*(u) &= \frac{\Sigma_{WW}^{-1}\Sigma_{WV}u}{\sqrt{u^T\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WW}\Sigma_{WW}^{-1}\Sigma_{WV}u}} \\ &= \frac{\Sigma_{WW}^{-1}\Sigma_{WV}u}{\sqrt{u^T\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u}} \end{aligned}$$

Plugging this back into the original objective function gives the following optimization problem for u.

Maximize $u^T\Sigma_{VW}v^*(u)$ for $u \in V^*$ such that $u^T\Sigma_{VV}u = 1$.

$$\begin{aligned} u^T\Sigma_{VW}v^*(u) &= \frac{u^t\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u}{\sqrt{u^T\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u}} \\ &= \sqrt{u^T\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u} \end{aligned}$$

Again we can take the gradient of the objective function and set it equal to a constant times the gradient of the constraint. This gives the following.

$$\begin{aligned} \frac{1}{2}\left(u^T\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u\right)^{-1/2} 2\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u &= 2\lambda\Sigma_{VV}u \\ \Sigma_{VV}^{-1}\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u &= \lambda' u \end{aligned}$$

In other words u must be an eigenvector of $\Sigma_{VV}^{-1}\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV} : V^* \to V^*$. Furthermore we can show $\lambda' \geq 0$ as follows where we use the fact that the inverse of a symmetric positive semi-definite abstract matrix is also positive semidefinite and hence Σ_{WW}^{-1} is positive semidefinite.

$$\begin{aligned} \Sigma_{VV}^{-1}\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u &= \lambda' u \\ \Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u &= \lambda'\Sigma_{VV}u \\ u^T\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}u &= \lambda' u^T\Sigma_{WW}u \\ \lambda' &= \frac{\left(\Sigma_{WV}u\right)^T\Sigma_{WW}^{-1}\left(\Sigma_{WV}u\right)}{u^T\Sigma_{VV}u} \\ &\geq 0 \end{aligned}$$

$$\begin{aligned}
\rho_{u,v^*(u)} &= \frac{u^T \Sigma_{VW} \left(\Sigma_{WW}^{-1} \Sigma_{WV} u\right)}{\sqrt{u^T \Sigma_{VV} u} \sqrt{\left(u^T \Sigma_{VW} \Sigma_{WW}^{-1}\right) \Sigma_{WW} \left(\Sigma_{WW}^{-1} \Sigma_{WV} u\right)}} \\
&= \frac{\sqrt{u^T \Sigma_{VW} \Sigma_{WW}^{-1} \Sigma_{WV} u}}{\sqrt{u^T \Sigma_{VV} u}} \\
&= \frac{\sqrt{u^T \Sigma_{VV} \left(\Sigma_{VV}^{-1} \Sigma_{VW} \Sigma_{WW}^{-1} \Sigma_{WV} u\right)}}{\sqrt{u^T \Sigma_{VV} u}} \\
&= \frac{\sqrt{u^T \Sigma_{VV} (\lambda' u)}}{\sqrt{u^T \Sigma_{VV} u}} \\
&= \sqrt{\lambda'}
\end{aligned}$$

In other words, for u an eigenvector of $\Sigma_{VV}^{-1} \Sigma_{VW} \Sigma_{WW}^{-1} \Sigma_{WV}$ we have that the correlation $\rho_{u,v^*(u)}$ is equal to the square root of the eigenvalue λ'. So we can take (u_1, v_1) to be the pair $(u, v^*(u))$ where u is the eigenvector of largest eigenvalue. Furthermore, the abstract matrix $\Sigma_{VV}^{-1} \Sigma_{VW} \Sigma_{WW}^{-1} \Sigma_{WV}$ is of the form $DM : V^* \to V^*$ where $D : V \times V \to R$ is the inner product Σ_{VV}^{-1} on V and $M : V^* \times V^* \to R$ is the symmetric abstract matrix $\Sigma_{VW} \Sigma_{WW}^{-1} \Sigma_{WV}$. The eigenvectors $u : V^*$ of DM are orthogonal as measured by D^{-1} which is Σ_{VV} in this case. So if we take u_2 to be the second strongest eigenvector of DM then we get $u_2^T \Sigma_{VV} u_1 = 0$ as required. More generally we can take u_1, ..., u_k to be the k strongest eigenvectors of $\Sigma_{VV}^{-1} \Sigma_{VW} \Sigma_{WW}^{-1} \Sigma_{WV}$.

Problem 17.1: Consider two vector spaces V and W and a probability distribution on pairs (x, y) with $x \in V$ and $y \in W$ with $\mathrm{E}[x] = 0$ and $\mathrm{E}[y] = 0$. Consider the following generalized matrices.

$$\begin{aligned}
\Sigma_{VV} &= \mathrm{E}\left[x x^\top\right] \\
\Sigma_{WW} &= \mathrm{E}\left[y y^\top\right] \\
\Sigma_{VW} &= \Sigma_{W,V}^\top = \mathrm{E}\left[x y^\top\right]
\end{aligned}$$

a. Give types for each of these abstract matrices both as bilinear functions on the reals and as functions between vector spaces.

b. Give the type of $\Sigma_{V,W} \Sigma_{WW}^{-1} \Sigma_{WV}$.

c. Use these abstract matrices to define two different inner products on V and describe how this relates to problem 16.2 and to CCA.

17.3 Voldemort Challenges CCA

In practice we do not have access to the true expectations and must work with a finite sample $\langle x_1, y_1 \rangle$, ..., $\langle x_n, y_n \rangle$. In the case where $n < d$ Voldemort can defeat CCA — there is no way to name even a single feature defined on all of V or all of W without introducing some other object, such as an inner product, to break the symmetries in the vector spaces.

Chapter 18

Kernels and Hilbert Spaces

Kernel methods provide "nonlinear" generalizations of algorithms using ambient inner products. For example, we have kernelized gradient descent (in a function space), kernelized ridge regression and kernelized PCA. Other algorithms, not discussed here, can also be kernelized. For example we have kernelized support vector machines and kernelized independent component analysis. A kernel, as we will see below, is a special kind of similarity function representing an inner product.

18.1 Function Spaces and $\mathcal{F}(K)$

In many cases we are interested in a space of functions $f : \mathcal{X} \to \mathcal{R}$ where $\mathcal{X}$ is an arbitrary "input space". This is particularly true in machine learning where we often want to take a set of input-output pairs $\langle x_1, y_1 \rangle$, ...,$\langle x_N, y_N \rangle$ with $x_i : \mathcal{X}$ and $y_i : \mathcal{R}$ and generalizing the given pairs to a function $f : \mathcal{X} \to \mathcal{R}$. We define the type **FunSpace** as follows.

$$\mathbf{FunSpace} \equiv \overline{\mathcal{X} : \mathbf{type}_1;\ \mathcal{F} : \mathbf{SetOf}(\mathcal{X} \to \mathcal{R});\ \mathcal{A}}$$

The axioms $\mathcal{A}$ state that $\mathcal{F}$ is closed under addition and scaling:

- The family $\mathcal{F}$ is closed under function addition — for $f \in \mathcal{F}$ and $g \in \mathcal{F}$ then we have $f + g \in \mathcal{F}$ where $f + g$ is the function defined by $(f + g)(x) = f(x) + g(x)$.
- The family $\mathcal{F}$ is closed under scaling — for $f \in \mathcal{F}$ and a constant $a : \mathcal{R}$ we have $af \in \mathcal{F}$ where af is the function defined by $(af)(x) = af(x)$.

One can show that a function space under function addition and scaling forms a vector space. For a function space $\mathcal{F}$ we will write $V(\mathcal{F})$ for the vector space consisting of the functions in $\mathcal{F}$ under function addition and function scaling. Function space have more structure than vector spaces in the sense that we can have distinct (non-isomorphic) function spaces $\mathcal{F}_1$ and $\mathcal{F}_2$ such

that the vector spaces $V(\mathcal{F}_1)$ is isomorphic (as a vector space) to $V(\mathcal{F}_2)$. For example $\mathcal{F}_1$ and $\mathcal{F}_2$ can both one-dimensional (as a vector space) but where $\mathcal{F}_1$ contains only constant functions while $\mathcal{F}_2$ contains no constant functions.

For a given input space $\mathcal{X}$ we define a similarity function on $\mathcal{X}$ to be any $K : \mathcal{X} \times \mathcal{X} \to \mathcal{R}$ with $K(x_1, x_2) \geq 0$. Given an input space $\mathcal{X}$, a similarity function $K : \mathcal{X} \times \mathcal{X} \to \mathcal{R}$, and an input $x : \mathcal{X}$ we define $K_x : \mathcal{X} \to \mathcal{R}$ to be the function defined by $K_x(y) = K(x, y)$. We define $\mathcal{F}(K)$ to be the function space containing all functions of the form $\sum_{i=1}^{N} \alpha_i K_{x_i}$ with $\alpha_i : \mathcal{R}$. For any similarity function K we have that $\mathcal{F}(K)$ is a function space (and hence a vector space). For $f = \sum_{i=1}^{N} \alpha_i K_{x_i}$ we have that $f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x)$.

In machine learning applications one often wants to take a set of input-output pairs $\langle x_1, y_1 \rangle$, ...,$\langle x_N, y_N \rangle$ with $x_i : \mathcal{X}$ and $y_i : \mathcal{R}$ and generalizing the given pairs to a function $f : \mathcal{X} \to \mathcal{R}$. It seems natural to consider a similarity function K and generalize to a function of the form $f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x)$ where the x_i are the inputs appearing in the training data. One them makes a prediction based on the similarity of a new input to inputs in the training data. However, it is not clear how to set the coefficients α_i. In the following sections we define a special kind of similarity function, a kernel, such that $K(x_1, x_2)$ represents an inner product in a vector space. When the kernel represents an inner product we can apply standard vector space algorithms, such as ridge regression, which exploit an inner product operation.

18.2 Hilbert Spaces and ℓ_2

An inner product space is a vector space together with an inner product on that space. A Hilbert space is a complete inner product space — an inner product space such that every Cauchy sequence of vectors has a limit. To explain the completeness requirement more fully we first define the concept of an epsilon ball. For a Hilbert space H with inner product D and for a vector $x : H$ we will write $||x||$ for $\sqrt{x^\intercal D x}$ and for a real number $\epsilon > 0$ we define the ϵ-ball around x, denoted $B_\epsilon(x)$ to be the set of vectors $y : H$ with $||x - y|| \leq \epsilon$. A Cauchy sequence in H is an infinite sequence x_1, x_2, x_3, ... such that for all $\epsilon > 0$ there exists k such that the ball $B_\epsilon(x_k)$ contains all vectors x_j for $j > k$.

> **Problem** 18.1: Show that all finite dimensional vector spaces are complete. Hint: pick an orthonormal coordinate system and use the fact that the ith coordinate of a Cauchy sequence of vectors is a Cauchy sequence of reals. Use the fact that every Cauchy sequence of reals has a limit.

The above problem shows that the distinction between an inner product space and a Hilbert space is only meaningful in infinite dimension. For this reason one generally only considers infinite dimensional Hilbert spaces — a finite dimensional Hilbert space should be called an inner product space.

As an example of a Hilbert space we consider the set ℓ_2 of all infinite sequences x^1, x^2, ... with x^i of real number which are square-summable in the

sense that $\sum_{i=1}^{\infty}(x^i)^2 < \infty$. We can add sequences and multiply a sequence by a scalar in the obvious way and these operations on the square-summable sequences form a vector space. We define an inner product D_2 on sequences in ℓ_2 by the following equation.

$$x^\intercal D_2 y = \langle x^1, x^2, \ldots \rangle \, D_2 \, \langle y^1, y^2, \ldots \rangle = x^1 y^1 + x^2 y^2 + \cdots$$

We can show that the infinite sum on the right hand side is well defined as follows where we write $||x||_2^2$ for $\sum_{i=1}^{\infty}(x^i)^2$.

$$\begin{aligned} \sum_i |x^i y^i| &\leq \sum_i (x^i)^2 + (y^i)^2 \\ &= ||x||_2^2 + ||y||_2^2 \\ &< \infty \end{aligned}$$

It is straightforward to check that D_2 is bilinear, symmetric and positive definite and hence is an inner product. Since we have that D_2 is an inner product on a vector space we immediately have the Cauchy-Schwarz inequality.

$$x^\intercal D_2 y \leq ||x||_2 \, ||y||_2$$

Problem 18.2: Show that ℓ_2 is complete — that every Cauchy sequence has a limit. Hint: Show that if $x_1,\ x_2, x_3, \ldots$ is a Cauchy sequence of elements of ℓ_2 then for each coordinate j we have that $x_1^j, x_2^j, x_3^j, \ldots$ is a Cauchy sequence of reals.

Problem 18.3: Give an example of an infinite dimensional vector space, and an inner product on that space, which is not complete. Hint: Consider the subspace of ℓ_2 consisting of the eventually zero sequences — the sequences $x^1, x^2, x^3, \ldots$ such that there exists a k for which $x^j = 0$ for all $j > k$. Give an example of a Cauchy sequence of vectors in this subspace whose limit is not in this subspace.

Now consider a Hilbert space H for which no finite basis exists. A countable basis for H is an infinite sequence e_1, e_2, e_3, ... such that any finite prefix e_1, ..., e_k is linearly independent and the infinite sequence spans the space in the sense that for any vector x in H there exists real coefficients a_1, a_2, a_3, ... such that $x = \sum_i a_i e_i$, or more rigorously, that the vector sequence $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$, ... defined by $\hat{x}_n = \sum_{i=1}^{n} a_i e_i$ is a Cauchy sequence converging to x. The space ℓ_2 has the obvious countable basis.

A Hilbert space for which there exists a countable basis is called separable. For a finite dimensional vector space V with dimension d we have that V is isomorphic to $\mathcal{R}^d$ — the vector space whose elements are d-tuples of real numbers. Similarly every separable Hilbert space is isomorphic to ℓ_2. One can show this by running an infinite version of the Gram-Schmidt procedure. In particular let H be a Hilbert space and let e_1, e_2, e_3, ... be a countable basis

for H. We can construct an orthonormal basis e'_1, e'_2, e'_3 ... by first letting e'_1 be the normalization of e_1 and then computing e'_{j+1} to be the normalization of $e_{j+1} - \sum_{i=1}^{j} e_{j+1}^{\intercal} D e'_i$. For an orthonormal basis $B = e_1, e_2, \ldots$ one can show show that any vector x has unique coordinates $a_1, a_2, \ldots$ such that $x = \sum_{i=1}^{\infty} a_i e_i$. The mapping from x to its coordinates under an orthonormal basis defines an isomorphism from H to ℓ_2.

18.3 Kernels

We now consider an arbitrary input space (type) $\mathcal{X}$ and a function $K : \mathcal{X} \times \mathcal{X} \to \mathcal{R}$. We call K a Kernel on $\mathcal{X}$ if there exists a Hilbert space H and a function (a feature map) $\Phi : \mathcal{X} \to H$ such that for any $x_1, x_2 : \mathcal{X}$ we have $K(x_1, x_2) = \Phi(x_1)^{\intercal}(H.D)\Phi(x_2)$ where $H.D$ is the inner product of H. If the Hilbert space H can be taken to be separable then we say that K is a separable kernel.

> **Problem** 18.4: Use the inference rule of substitution of isomorphics to argue that K is a separable kernel if and only if there exists a feature map $\Phi : \mathcal{X} \to \ell_2$ such that for $x_1, x_2 : \mathcal{X}$ we have $K(x_1, x_2) = \Phi(x_1)^{\intercal} D_2 \Phi(x_2)$.

We will now give methods for constructing separable kernel functions. We will prove the following.

1. If V is a finite dimensional vector space over the reals and D is an inner product on V then $D : V \times V \to \mathcal{R}$ is a separable kernel on V.

   proof: Here we are taking the input space to be the vector space and we can take the feature map to be the identity map.

2. If K_1 is a separable kernel on an input space $\mathcal{Y}$ then for $f : \mathcal{X} \to \mathcal{Y}$ the kernel K_2 defined by $K_2(x_1, x_2) = K_1(f(x_1), f(x_2))$ is a separable kernel on $\mathcal{X}$.

   proof: Let $\Phi_1 : \mathcal{Y} \to \ell_2$ be a feature map for K_1. We can define a feature map Φ_2 for K_2 by $\Phi_2^i(x) = \Phi_1^i(f(x))$.

3. For $f : \mathcal{X} \to \mathcal{R}$ we have that $K(x_1, x_2) = f(x_1)f(x_2)$ is a separable kernel on $\mathcal{X}$.

   proof: We can define a feature map $\Phi : \mathcal{X} \to \mathcal{R}$ by taking the first feature $\Phi^1(x) = f(x)$ and for features $j > 1$ we take $\Phi^j(x) = 0$.

4. For any separable kernel K_1 on $\mathcal{X}$ and real number $a > 0$ we have that $K_2(x_1, x_2) = aK_1(x_1, x_2)$ is a separable kernel on $\mathcal{X}$

   Proof: If Φ is a feature map for K_1 then $a\Phi$ is a feature map for K_2. Note that $a > 0$ is required for K_2 to be positive definite.

5. For any separable kernels K_1 and K_2 on $\mathcal{X}$ we have that $K_3(x_1, x_2) = K_1(x_1, x_2) + K_2(x_1, x_2)$ is a separable kernel on $\mathcal{X}$.

   Proof. Let $\Phi_1 : \mathcal{X} \to \ell_2$ and $\Phi_2 : \mathcal{X} \to \ell_2$ be feature maps for K_1 and K_2. For an input $x : \mathcal{X}$ we then have features of the form $\Phi_1^i(x)$ and $\Phi_2^j(x)$. This is a countable number of features and hence the collection of all these features can be represented by a single map $\Phi_3 : \mathcal{X} \to \mathcal{R}$.

6. For any separable kernels K_1 and K_2 on $\mathcal{X}$ we have that $K_3(x_1, x_2) = K_1(x_1, x_2)K_2(x_1, x_2)$ is a separable kernel on $\mathcal{X}$.

   proof: Let $\Phi_1 : \mathcal{X} \to \ell_2$ and $\Phi_2 : \mathcal{X} \to \ell_2$ be feature maps for K_1 and K_2 respectively. We then have the following.

$$\begin{aligned} K_1(x_1, x_2)K_2(x_1, x_2) &= \left(\sum_{i=1}^{\infty} \Phi_1^i(x_1)\Phi_1^i(x_2)\right)\left(\sum_{i=1}^{\infty} \Phi_2^j(x_1)\Phi_2^j(x_2)\right) \\ &= \sum_{i,j} \left(\Phi_1^i(x_1)\Phi_1^i(x_2)\right)\ \left(\Phi_2^j(x_1)\Phi_2^j(x_2)\right) \\ &= \sum_{i,j} \left(\Phi_1^i(x_1)\Phi_2^j(x_1)\right)\ \left(\Phi_1^i(x_2)\Phi_2^j(x_2)\right) \end{aligned}$$

   We can then take the feature map Φ_3 for K_3 to be defined by $\Phi_3^{i,j}(x) = \Phi_1^i(x)\Phi_2^j(x)$. Note that Φ_3 has a countable set of features. We omit the proof that the features in $\Phi_3(x)$ are square-summable.

7. For an infinite sequence $K_1, K_2, \ldots$ of separable kernels on $\mathcal{X}$ such that for any $x : \mathcal{X}$ we have that $\sum_{i=1}^{\infty} K_i(x, x)$ is finite we have that $K(x_1, x_2) = \sum_{1=1}^{\infty} K_i(x_1, x_2)$ is a separable kernel.

   proof: Let $\Phi_i : \mathcal{X} \to \ell_2$ be a feature map for K_i. We can define a feature map for K by $\Phi^{i,j}(x) = \Phi_i^j(x)$. Again note that Φ has a countable number of features. We omit the proof that the condition $\sum_k K_k(x, x) < \infty$ implies that the features of $\Phi(x)$ are square-summable.

For any vector space V and inner product Σ on V^* we consider the Gaussian kernel on V with covariance Σ defined as follows.

$$K(x_1, x_2) = e^{-\frac{1}{2}(x_1 - x_2)^\intercal \Sigma^{-1}(x_1 - x_2)}$$

We now show that the Gaussian kernel is a separable kernel. First we observe the following.

$$K(x_1, x_2) = \left(e^{-\frac{1}{2}x_1^\intercal \Sigma^{-1} x_1} e^{-\frac{1}{2}x_2^\intercal \Sigma^{-1} x_2}\right)\ \left(e^{x_1^\intercal \Sigma^{-1} x_2}\right)$$

The left hand term is a separable kernel as it only involves a single feature (kernel formation rule 3). Since a product of separable kernels is a separable kernel (formation rule 6), it suffices to show that the right hand term is a

separable kernel. We consider the power series for the exponential function. Each term in this series has the form $a_j K(x_1, x_2)^j$ with $a_j > 0$. Each term in the series is then a finite product of separable kernels multiplied by a positive constant and is therefore a separable kernel. Finally we have that an infinite sum of separable kernels is a separable kernel provided that the sum always converges which it does in this case.

18.4 Reproducing Kernel Hilbert Space (RKHS)

We now define the notion of a reproducing kernel Hilbert space (RKHS) and show that if K is a separable kernel then the completion of the function space $\mathcal{F}(K)$ is an RKHS. Let $\Phi : \mathcal{X} \to \ell_2$ be a feature map for a separable kernel K on an input space $\mathcal{X}$. For $\omega \in \ell_2$ we define $f_\omega : \mathcal{X} \to \mathcal{R}$ by $f_\omega(x) = \omega^\intercal D_2 \Phi(x)$. Note that $f_{\Phi(x)} = K_x$. More explicitly we have $f_{\Phi(x_1)}(x_2) = \Phi(x_1)^\intercal D_2 \Phi(x_2) = K(x_1, x_2) = K_{x_1}(x_2)$. For $f \in \mathcal{F}(K)$ we define $\omega(f)$ as follows.

$$\omega(f) = \underset{f_\omega = f}{\operatorname{argmin}} \, ||\omega||_2^2$$

We first show that this definition yields the following.

$$\omega(K_x) = \Phi(x)$$

To see this suppose $\omega(K_x) \neq \Phi(x)$ and consider $\omega_\perp = \omega(K_x) - \Phi(x)$. We then have that $f_{\omega(K_x)} = K_x = f_{\Phi(x)}$ which implies that $f_{\omega_\perp}$ is the zero function. In particular we have that $f_{\omega_\perp}(x) = 0$ which implies that $\omega_\perp$ is orthogonal to $\Phi(x)$. We now have $\omega(K_x) = \omega_\perp + \Phi(x)$ and hence $||\omega(K_x)||_2^2 = ||\omega_\perp||_2^2 + ||\Phi(x)||_2^2 > ||\Phi(x)||_2^2$ and we have a contradiction. A similar argument can be used to show the following.

$$\omega\left(\sum_{i=1}^{N} \alpha_i K_{x_i}\right) = \sum_{i=1}^{n} \alpha_i \Phi(x_i)$$

This shows that for f in $\mathcal{F}(D)$ we have that $\omega(f)$ is well defined — the optimization problem defining $\omega(f)$ has a unique minimum. We now define the inner product D_K of two function f and g in $\mathcal{F}(K)$ as follows.

$$f^\intercal D_K g = \omega(f)^\intercal D_2 \omega(g)$$

The inner product of functions is traditionally written as $\langle f, g \rangle$ but we will continue to write it in terms of the inner product operation D_K. We now have the following.

$$K_{x_1}^\intercal D_K K_{x_2} = \Phi(x_1)^\intercal D_2 \Phi(x_2) = K(x_1, x_2)$$

$$\left(\sum_{i=1}^{N} \alpha_i K_{x_i}\right) D_K \left(\sum_{j=1}^{M} \alpha_j K_{x_j}\right) = \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)$$

This second equation implies that, for f and g in $\mathcal{F}(K)$, the inner product $f^\intercal D_K g$ depends only on the Kernel K and is independent of the choice of the feature map Φ. The first equation above is called the reproducing property of an RKHS — the inner product on the function space reproduces the kernel on the input space. The following equation is the evaluation property.

$$f^\intercal D_K K_x = \omega(f)^\intercal D_2 \Phi(x) = f(x)$$

This shows that K_x can be viewed as a kind of generalized Dirac delta function for which we have $\langle f, \delta_x \rangle = \int_{-\infty}^{\infty} f(y)\delta_x(y)dy = f(x)$.

Since we now have an inner product defined on the functions in $\mathcal{F}(K)$ we have a well defined notion of a Cauchy sequence of functions and can complete the space by taking limits of Cauchy sequences. Given a feature map we construct the limit of a function sequence f_1, f_2, ... by considering the feature vector sequence $\omega(f_1)$, $\omega(f_2)$, ... and taking the function limit to be f_ω where ω is the feature vector limit. The complete function space can be defined as the set of functions of the form f_ω for $\omega \in \ell_2$. A separable RKHS can be defined to be a Hilbert space of functions which can be written as the completion of $\mathcal{F}(K)$ for some separable kernel K.

An RKHS can also be defined as a Hilbert space H of functions from an input space $\mathcal{X}$ to the reals (or complex numbers) such that for each input x the map $L_x : H \to \mathcal{R}$ defined by $L_x(f) = f(x)$ is continuous. It turns out that in this case one can construct a kernel K such that H is the completion of $\mathcal{F}(K)$.

18.5 Fourier Analysis and Feature Maps

18.6 Symmetry (and Voldemort) in Hilbert Space

Chapter 19

Banach Spaces

19.1 A Norm on a Vector Space

Consider a vector space V over the reals. A norm on V is a function $N : V \to \mathcal{R}$ satisfying the following conditions for $x, y : V$.

- **Positivity:** $N(x) \geq 0$ and $N(x) = 0$ if and only if $x = 0$.
- **Scale Invariance:** For $a : \mathcal{R}$ we have

$$N(ax) = |a|N(x)$$

- **Sub-Additivity or Triangle Inequality:**

$$N(x + y) \leq N(x) + N(y)$$

It is worth noting that $N(x) \geq 0$ follows from $N(0) = 0$, scale invariance, and the triangle inequality. More specifically, given $N(0) = 0$ we have $0 = N(x - x) \leq N(x) + N(x) = 2N(x)$.

19.2 Banach Spaces

A Banach space is a complete normed vector space — a vector space together with a norm on that space such that every Cauchy sequence has a limit. Here a Cauchy sequence is defined to be an infinite sequence of vectors x_1, x_2, x_3, ... such that for every real number $\epsilon > 0$ there exists k such that for all $j > k$ we have $x_j \in B_\epsilon(x_k)$ where $B_\epsilon(x)$ is defined by the norm N as follows.

$$B_\epsilon(x) = \{y : V \mid \ N(y - x) \leq \epsilon\}$$

For any Hilbert space H with inner product D we have that $N(x) = \sqrt{x^\intercal D x}$ is a norm on H and the vectors of H under this norm form a Banach space.

Recall that a Hilbert space is separable if there exists a countable basis. All separable Hilbert spaces are isomorphic to ℓ_2 — up to isomorphism there is only one separable Hilbert Space. The situation is different for Banach spaces. For $p > 0$ we can define ℓ_p to be the set of infinite sequences of real number x_1, x_2, ... such that $\sum_{i=1}^{\infty} |x_i|^p < \infty$. It is possible to show that the p-norm defined by

$$L_p(x) = \left(\sum_{i=1}^{\infty} x_i^p \right)^{1/p}$$

is a norm on ℓ_p and that ℓ_p under this norm is complete. It is possible to show that the resulting Banach space ℓ_1 is not isomorphic to the Banach space ℓ_2 and in fact there are many (non-isomorphic) separable Banach spaces.

We can similarly define $\mathcal{L}_p$ to be the set of functions $f : \mathcal{R} \to \mathcal{R}$ such that $\int_{-\infty}^{\infty} |f(x)|^p dx < \infty$. We can then define the norm L_p on $\mathcal{L}_p$ as follows.

$$L_p(f) = \left(\int_{-\infty}^{\infty} |f(x)|^p dx \right)^{\frac{1}{p}}$$

It can be shown that $\mathcal{L}_p$ under this norm is complete and hence forms a Banach space.

19.3 Dual Norms

Consider a normed vector space V with norm N. The dual norm N^* is defined as follows on a dual vector ω in V^*.

$$N^*(\omega) = \sup_{N(x)=1} \omega^\intercal x$$

We will now that the dual norm is a norm.

19.4 p-q Norms

Given a basis $B = \langle we_1, \ldots, e_d \rangle$ and a real number $p > 0$ define $||x||_p$ by $||x||_p = \left(\sum_{i=1}^{d} (a_i^B(x)^p \right)^{1/p}$. We will work in a fixed basis and write

The dual norm $N_p^*(\omega)$ can be computed as follows. We maximize $\sum_i \omega_i x_i$ subject to the constraint $\sum_i x_i^p = 1$. Setting the gradient of the objective equal to λ times the gradient of the constraint yields the following.

$$\begin{aligned} \omega_i &= \lambda p x_i^{p-1} \\ x_i &= \left(\frac{1}{\lambda p} \right)^{\frac{1}{p-1}} \omega_i^{\frac{1}{p-1}} \\ &= \lambda' \omega_i^{\frac{1}{p-1}} \end{aligned}$$

We can now solve for λ' by substituting this back into the constraint $\sum_i x_i^p = 1$

$$\begin{aligned}\sum_i \left(\lambda' \omega_i^{\frac{1}{p-1}}\right)^p &= 1 \\ \lambda' &= \left(\sum_i \omega_i^{\frac{p}{p-1}}\right)^{-\frac{1}{p}}\end{aligned}$$

We can now compute the dual norm of ω as follows.

$$\begin{aligned}N_p^*(\omega) &= \sum_i \omega_i x_i \\ &= \sum_i \omega_i \lambda' \omega_i^{\frac{1}{p-1}} \\ &= \lambda' \sum_i \omega_i^{\frac{p}{p-1}} \\ &= \left(\sum_i \omega_i^{\frac{p}{p-1}}\right)^{-\frac{1}{p}} \left(\sum_i \omega_i^{\frac{p}{p-1}}\right) \\ &= \left(\sum_i \omega_i^{\frac{p}{p-1}}\right)^{\frac{p-1}{p}} \\ &= N_q(\omega) \quad \text{for } q = \frac{p}{p-1} \text{ or equivalently } \frac{1}{p} + \frac{1}{q} = 1\end{aligned}$$

19.5 Voldemort in Banach Space

Chapter 20

Fisher Information

Consider a vector space V and a doubly differentiable map P from V to probability distributions on a space $\mathcal{Y}$. For $\Theta \in V$ We will write P_Θ for the distribution that P associates with Θ. For convenience we will assume that $\mathcal{Y}$ is countable but all the results here can be extended to the case where $\mathcal{Y}$ is continuous and P_Θ is atomless (no point in $\mathcal{Y}$ ever has probability greater than zero).

We define the Fisher information "matrix" (bilinear function) as follows.

$$I[\Theta] = (\nabla_\Psi \nabla_\Psi KL(P_\Theta, P_\Psi))[\Psi \leftarrow \Theta]$$

$$\begin{aligned} KL(P_\Theta, P_\Psi) &= \sum_y P_\Theta(y) \ln \frac{P_\Theta(y)}{P_\Psi(y)} \\ &= \left(\sum_y P_\Theta(y) \ln \frac{1}{P_\Psi(y)} \right) - H(P_\Theta) \end{aligned}$$

$$\begin{aligned} \nabla_\Psi KL(P_\Theta, P_\Psi) &= \sum_y -P_\Theta(y) \nabla_\psi \ln P_\Psi(y) \\ &= -\sum_y P_\Theta(y) \frac{1}{P_\Psi(y)} \nabla_\Psi P_\Psi(y) \end{aligned}$$

$$\begin{aligned} (\nabla_\Psi KL(P_\Theta, P_\Psi))[\Psi \leftarrow \Theta] &= -\sum_y \frac{P_\Theta(y)}{P_\Theta(y)} (\nabla_\Theta P_\Theta(y)) \\ &= -\nabla_\Theta \sum_y P_\Theta(y) = -\nabla_\Theta 1 = 0 \end{aligned}$$

$$\nabla_\Psi \nabla_\Psi KL(P_\Theta, P_\Psi) = \sum_y P_\Theta(y) \left(\frac{1}{P_\Psi(y)^2} (\nabla_\Psi P_\Psi(y)) (\nabla_\Psi P_\Psi(y)) - \frac{1}{P_\Psi(y)} \nabla_\Psi \nabla_\Psi P_\Psi(y) \right)$$

$$\begin{aligned}
\nabla_\Psi \nabla_\Psi KL(P_\Theta, P_\Psi)[\Psi \leftarrow \Theta] &= \sum_y P_\Theta(y) \left(\frac{1}{P_\Theta(y)^2} \left(\nabla_\Psi P_\Theta(y)\right) \left(\nabla_\Theta P_\Theta(y)\right) \right) \\
I[\Theta] &= \mathrm{E}_{y \sim P_\Theta} \left[\left(\nabla_\Theta \ln P_\Theta(y)\right) \left(\nabla_\Theta \ln P_\Theta(y)\right)^\intercal \right]
\end{aligned}$$

Chapter 21

Manifolds

21.1 Topological Manifolds

The type of topological spaces can be defined as follows.

$$\mathbf{TopSpace} \equiv \overline{\alpha{:}\mathbf{type}_1;\ \mathcal{O}{:}\mathbf{SetOf}(\mathbf{SetOf}(\alpha));\ \mathcal{A}}$$

The sets in $\mathcal{O}$ are called the open sets. The conditions $\mathcal{A}$ state that any finite intersection of open sets is also open and that any (possibly uncountably infinite) union of open sets is open.

For any d-dimensional vector space V we can associate V with a topology by considering the open boxes defined as follows where $e_1, \ldots e_d$ forms basis of V.

$$\mathbf{Box}(x, e_1, \ldots, e_d) = \{y \mid y = x + a_1 e_1 + \cdots + a_n e_n \text{ with } 0 < a_i < 1\}$$

We can then take the topology of V to be generated by all such boxes over all choices of the basis vectors. More specifically, we take the collection of open sets to be the least collection containing these boxes and that is closed under finite intersections and arbitrary unions.

For any Banach space B we can construct a topology (a family of open sets) on B by consider the open balls of the form

$$B_\epsilon(x) = \{y \mid N(x - y) < \epsilon\}$$

We can get a topology by closing under finite intersection and arbitrary union. For a finite dimensional space the topology is independent of the choice of the norm — any finite dimensional vector space can be assigned a topology

Open sets containing a point x are often called neighborhoods of x. A point x is called a limit point of a set S if every neighborhood of x intersects S. For any set S we define the closure of S, denoted $\overline{S}$, to be S plus all limit points of S. One can prove that in an arbitrary topological space and an arbitrary subset S of that space we have that the complement of $\overline{S}$ is open. To see this we note

that for $y \notin \overline{S}$ there must exist a neighborhood U_y of y which does not intersect S. We also have that U_y also does not contain any limit points of S because for $x \in U_y$ we have that U_y is a neighborhood of x not intersecting S so x is also not a limit point of S. The complement of $\overline{S}$ is then the union of all such open sets U_y for y in th complement of $\overline{S}$.

For any topological space X and subset S of X we can consider the family of subsets of S of the form $S \cup U$ where U is an open set of X. It is possible to that this collection of subsets of S is closed under finite intersection and union and hence forms a topology on S called the subset topology.

An open interval in $\mathcal{R}$ under the subset topology is isomorphic (as a topological space) to the entire set of reals $\mathcal{R}$. Similarly, an open ball in a finite dimensional vector space is isomorphic (as a topological space) to the entire vector space.

A topological space is called a d-manifold if for every point x there exists a neighborhood U of x such that U under the subset topology is isomorphic (as a topological space) to an open ball in a d-dimensional vector space (which is isomorphic to the entire d-dimensional vector space).

A topological space is called compact if every open cover (every collection of open sets whose union is all of X) has a finite subcover.

For two topological space X and Y and a function $f : X \to Y$, we say that f is continuous if for every point x of X and every neighborhood U of $f(x)$ there exists a neighborhood V of x such that $f(V) \subseteq U$.

For any topological space X and points x and y in X we define a path from x to y to be a continuous function $f : [0, 1] \to X$ with $f(0) = x$ and $f(1) = y$. A loop is a path from a point x to itself.

Jordan Curve theorem.

topological Space

21.2 Voldemort Challenges Topology

21.3 Smooth (Differentiable) Manifolds

21.4 Voldemort Challenges Smoothness

21.5 Riemannian Manifolds

21.6 Information Geometry

21.7 Natural Gradient Descent

21.8 Jeffery's Prior