Set Theory in Computer Science

A Gentle Introduction to Mathematical Modeling I

José Meseguer
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

© José Meseguer, 2008–2010; all rights reserved.

February 9, 2012

Contents

1 Motivation **7**

2 Set Theory as an Axiomatic Theory **11**

3 The Empty Set, Extensionality, and Separation **15**
3.1 The Empty Set . . . 15
3.2 Extensionality . . . 15
3.3 The Failed Attempt of Comprehension . . . 16
3.4 Separation . . . 17

4 Pairing, Unions, Powersets, and Infinity **19**
4.1 Pairing . . . 19
4.2 Unions . . . 21
4.3 Powersets . . . 24
4.4 Infinity . . . 26

5 Case Study: A Computable Model of Hereditarily Finite Sets **29**
5.1 HF-Sets in Maude . . . 30
5.2 Terms, Equations, and Term Rewriting . . . 33
5.3 Confluence, Termination, and Sufficient Completeness . . . 36
5.4 A Computable Model of HF-Sets . . . 39
5.5 HF-Sets as a Universe for Finitary Mathematics . . . 43
5.6 HF-Sets with Atoms . . . 47

6 Relations, Functions, and Function Sets **51**
6.1 Relations and Functions . . . 51
6.2 Formula, Assignment, and Lambda Notations . . . 52
6.3 Images . . . 54
6.4 Composing Relations and Functions . . . 56
6.5 Abstract Products and Disjoint Unions . . . 59
6.6 Relating Function Sets . . . 62

7 Simple and Primitive Recursion, and the Peano Axioms **65**
7.1 Simple Recursion . . . 65
7.2 Primitive Recursion . . . 67
7.3 The Peano Axioms . . . 69

8 Case Study: The Peano Language **71**

9 Binary Relations on a Set **73**
9.1 Directed and Undirected Graphs . . . 73
9.2 Transition Systems and Automata . . . 75
9.3 Relation Homomorphisms and Simulations . . . 76
9.4 Orders . . . 78

9.5 Sups and Infs, Complete Posets, Lattices, and Fixpoints . . . 81
9.6 Equivalence Relations and Quotients . . . 84
9.7 Constructing $\mathbb{Z}$ and $\mathbb{Q}$ . . . 88

10 Case Study: Fixpoint Semantics of Recursive Functions and Lispy 91
10.1 Recursive Function Definitions in a Nutshell . . . 92
10.2 Fixpoint Semantics of Recursive Function Definitions . . . 93
10.3 The Lispy Programming Language . . . 96

11 Sets Come in Different Sizes 97
11.1 Cantor's Theorem . . . 97
11.2 The Schroeder-Bernstein Theorem . . . 98

12 I-Indexed Sets 99
12.1 I-Indexed Sets *are* Surjective Functions . . . 99
12.2 Constructing I-Indexed Sets from other I-Indexed Sets . . . 104
12.3 I-Indexed Relations and Functions . . . 105

13 From I-Indexed Sets to Sets, and the Axiom of Choice 107
13.1 Some Constructions Associating a Set to an I-Indexed Set . . . 107
13.2 The Axiom of Choice . . . 112

14 Well-Founded Relations, and Well-Founded Induction and Recursion 117
14.1 Well-Founded Relations . . . 117
14.1.1 Constructing Well-Founded Relations . . . 118
14.2 Well-Founded Induction . . . 119
14.3 Well-Founded Recursion . . . 120
14.3.1 Examples of Well-Founded Recursion . . . 120
14.3.2 Well-Founded Recursive Definitions: Step Functions . . . 121
14.3.3 The Well-Founded Recursion Theorem . . . 123

15 Cardinal Numbers and Cardinal Arithmetic 125
15.1 Cardinal Arithmetic . . . 126
15.2 The Integers and the Rationals are Countable . . . 129
15.3 The Continuum and the Continuum Hypothesis . . . 131
15.3.1 Peano Curves . . . 132
15.3.2 The Continuum Hypothesis . . . 132

16 Classes, Intensional Relations and Functions, and Replacement 135
16.1 Classes . . . 135
16.1.1 Mathematical Theorems are Assertions about Classes . . . 138
16.2 Intensional Relations . . . 139
16.3 Intensional Functions . . . 141
16.3.1 Typing Intensional Functions . . . 142
16.3.2 Computing with Intensional Functions . . . 143
16.3.3 Dependent and Polymorphic Types . . . 144
16.4 The Axiom of Replacement . . . 145

17 Case Study: Dependent and Polymorphic Types in Maude 149
17.1 Dependent Types in Maude . . . 149
17.2 Polymorphic-and-Dependent Types in Maude . . . 152
17.3 Definability Issues . . . 154

18 Well Orders, Ordinals, Cardinals, and Transfinite Constructions **155**
18.1 Well-Ordered Sets . . . 155
18.2 Ordinals . . . 157
18.2.1 Ordinals as Transitive Sets . . . 159
18.2.2 Successor and Limit Ordinals . . . 159
18.2.3 Ordinal Arithmetic . . . 161
18.3 Transfinite Induction . . . 162
18.4 Transfinite Recursion . . . 163
18.4.1 α-Recursion . . . 164
18.4.2 Simple Intensional Recursion . . . 165
18.4.3 Transfinite Recursion . . . 166
18.5 Well-Orderings, Choice, and Cardinals . . . 168
18.5.1 Cardinals . . . 169
18.5.2 More Cardinal Arithmetic . . . 171
18.5.3 Regular, Singular, and Inaccessible Cardinals . . . 172

19 Well-Founded Sets and The Axiom of Foundation **173**
19.1 Well-Founded Sets from the Top Down . . . 173
19.1.1 $\ni$-Induction . . . 175
19.2 Well-Founded Sets from the Bottom Up . . . 176
19.3 The Axiom of Foundation . . . 178

Chapter 1

Motivation

"... we cannot improve the language of any science without at the same time improving the science itself; neither can we, on the other hand, improve a science, without improving the language or nomenclature which belongs to it."
(Lavoisier, 1790, quoted in Goldenfeld and Woese [23])

I found the inadequacy of language to be an obstacle; no matter how unwieldly the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain the precision that my purpose required. This deficiency led me to the idea of the present ideography. . . . I believe that I can best make the relation of my ideography to ordinary language clear if I compare it to that which the microscope has to the eye. Because of the range of its possible uses and the versatility with which it can adapt to the most diverse circumstances, the eye is far superior to the microscope. Considered as an optical instrument, to be sure, it exhibits many imperfections, which ordinarily remain unnoticed only on account of its intimate connection with our mental life. But, as soon as scientific goals demand great sharpness of resolution, the eye proves to be insufficient. The microscope, on the other hand, is prefectly suited to precisely such goals, but that is just why it is useless for all others.
(Frege, 1897, *Begriffsschrift*, in [52], 5–6)

Language and thought are related in a deep way. Without any language it may become impossible to conceive and express any thoughts. In ordinary life we use the different natural languages spoken on the planet. But natural language, although extremely flexible, can be highly ambiguous, and it is not at all well suited for science. Imagine, for example, the task of professionally developing quantum mechanics (itself relying on very abstract concepts, such as those in the mathematical language of operators in a Hilbert space) in ordinary English. Such a task would be virtually impossible; indeed, ridiculous: as preposterous as trying to build the Eiffel tower in the Sahara desert with blocks of vanilla ice cream. Even the task of *popularization*, that is, of explaining informally in ordinary English what quantum mechanics *is*, is highly nontrivial, and must of necessity remain to a considerable extent suggestive, metaphorical, and fraught with the possibility of gross misunderstandings.

The point is that without a precise scientific language it becomes virtually impossible, or at least enormously burdensome and awkward, to *think* scientifically. This is particularly true in mathematics. One of the crowning scientific achievements of the 20th century was the development of set theory as a precise language for all of mathematics, thanks to the efforts of Cantor, Dedekind, Frege, Peano, Russell and Whitehead, Zermelo, Fraenkel, Skolem, Hilbert, von Neumann, Gödel, Bernays, Cohen, and others. This achievement has been so important and definitive that it led David Hilbert to say, already in 1925, that "no one will drive us from the paradise which Cantor created for us" (see [52], 367–392, pg. 376). It was of

course possible to think mathematically before set theory, but in a considerably more awkward and quite restricted way, because the levels of generality, rigor and abstraction made possible by set theory are much greater than at any other previous time. In fact, many key mathematical concepts we now take for granted, such a those of an *abstract group* or a *topological space*, could only be formulated after set theory, precisely because the language needed to conceive and articulate those concepts was not available before.

Set theory is not really the only rigorous mathematical language. The languages of set theory and of mathematical logic were developed together, so that, as a mathematical discipline, set theory is a branch of mathematical logic. Technically, as we shall see shortly, we can view the language of set theory as a special sublanguage of *first-order logic*. Furthermore, other theories such as category theory and intuitionistic type theory have been proposed as alternatives to set theory to express all of mathematics.

There are various precise logical formalisms other than set theory which are particularly well-suited to express specific concepts in a given domain of thought. For example, temporal logic is quite well-suited to express properties satisfied by the dynamic behavior of a concurrent system; and both equational logic and the lambda calculus are very well suited to deal with functions and functional computation. However, set theory plays a privileged role as a mathematical language in which all the mathematical structures we need in order to give a precise meaning to the models described by various other logical languages, and to the satisfaction of formulas in such languages, can be defined.

All this has a direct bearing on the task of formal software specification and verification. Such a task would be meaningless, indeed utter nonsense and voodoo superstition, without the use of mathematical models and mathematical logic. And it is virtually impossible, or extremely awkward, to even *say* what needs to be said about such mathematical models and logical properties without a precise mathematical language. More importantly, it becomes virtually impossible to *think* properly without the conceptual tools provided by such a language. Either set theory or some comparable language become unavoidable: it is part of what any well educated computer scientist should be conversant with, like the air one breathes.

These notes include a review of basic set theory concepts that any well educated computer scientist should already be familiar with. Although they go beyond reviewing basic knowledge in various ways, nothing except basic acquaintance with the use of logical connectives and of universal and existential quantification in logic is assumed: the presentation is entirely self-contained, and many exercises are proposed to help the reader sharpen his/her understanding of the basic concepts. The exercises are an essential part of these notes, both because they are used in proofs of quite a few theorems, and because by solving problems in a mathematical theory one avoids having a superficial *illusion of understanding*, and gains real understanding. For those already well-versed in elementary set theory, these notes can be read rather quickly. However, some topics such as well-founded relations, well-founded induction, well-founded recursive functions, I-indexed sets, ordinals and cardinals, classes, and transfinite induction and recursion, may be less familiar. Also, already familiar notions are here presented in a precise, axiomatic way. This may help even some readers already thoroughly familiar with "naive" set theory gain a more detailed understanding of it as a logically axiomatized theory. Becoming used to reason correctly within an axiomatic theory —Euclidean geometry is the classical example, and axiomatic set theory follows the same conceptual pattern— is the best way I know of learning to think in a precise, mathematical way. Furthermore, a number of useful connections between set theory and computer science are made explicit in these notes; connections that are usually not developed in standard presentations of set theory.

I should add some final remarks on the style of these notes. There are three particular stylistic features that I would like to explain. First, these notes take the form of an extended *conversation* with the reader, in which I propose and discuss various problems, why they matter, and throw out ideas on how to solve such problems. This is because I believe that science itself is an ongoing critical *dialogue*, and asking questions in a probing way is the best way to understand anything. Second, I do not assume the proverbial *mathematical maturity* on the part of the reader, since such maturity is precisely the *quod erat demonstrandum*, and bringing it about is one of the main goals of these notes: I am convinced that in the 21st century mathematical maturity is virtually impossible without *mastering the language of set theory*. On the contrary, I assume the potential *mathematical immaturity* of some readers. This means that, particularly in the early chapters, there is a generous amount of what might be called mathematical spoon feeding, hand holding, and even a few nursery tales. This does not go on forever, since at each stage I *assume as known* all that has been already presented, that is, the mastery of the language already covered, so that in more advanced chapters, although the conversational style, examples, and motivation remain, the discourse gradually becomes

more mature. By the way, all this does not necessarily mean that the book is of no interest to mathematically mature readers, including professional mathematicians. What it does mean is that it should be an *easy read* for them; one from which they may gain at least two things: (i) a more reflective, systematic understanding of the set theory they use every day; and (ii) a good understanding of some of the ways in which mathematical models are used in computer science. The third stylistic feature I want to discuss is that the mindset of category theory, pervasive in modern mathematics, is present everywhere in these notes, but in Parts I and II this happens in a *subliminal* way. Categories and functors are only defined in Part III; but they are present from the beginning like a hidden music. And functorial constructions make early cameo appearances in Part I (very much like Alfred Hitchcock in his own movies) in four exercises and in §15. Indeed, §15, on cardinals and cardinal arithmetic, is a great beneficiary of this subliminal, categorical style; because much of basic cardinal arithmetic turns out to be an easy corollary of the functoriality of the set-theoretic constructions used to define arithmetic operations on cardinals. The tempo of this hidden categorical music picks up when universal algebra and first-order logic are discussed in Part II; but the rich harvest from this subliminal, categorical sowing is really reaped in Part III. In my view, this is a good way of gently preparing the reader, with a rich collection of examples, to fully appreciate the beauty, power, and conceptual economy of the language of category theory.

Chapter 2

Set Theory as an Axiomatic Theory

In mathematics all entities are studied following a very successful method, which goes back at least to Euclid, called the *axiomatic method*. The entities in question, for example, points, lines, and planes (or numbers, or real-valued functions, or vector spaces), are characterized by means of *axioms* that are postulated about them. Then one uses *logical deduction* to infer from those axioms the properties that the entities in question satisfy. Such properties, inferred from the basic axioms, are called *theorems*. The axioms, together with the theorems we can prove as their logical consequences, form a mathematical, axiomatic *theory*. It is in this sense that we speak of group theory, the theory of vector spaces, probability theory, recursion theory, the theory of differentiable real-valued functions, or set theory.

The way in which set theory is used as a language for mathematics is by expressing or translating other theories in terms of the theory of sets. In this way, everything can be *reduced* to sets and relations between sets. For example, a line can be precisely understood as a set of points satisfying certain properties. And points themselves (for example, in 3-dimensional space) can be precisely understood as triples (another kind of set) of real numbers (the point's coordinates). And real numbers themselves can also be precisely understood as sets of a different kind (for example as "Dedekind cuts"). In the end, all sets can be built out of the *empty* set, which has no elements. So all of mathematics can in this way be constructed, as it were, *ex nihilo*.

But sets themselves are also *mathematical* entities, which in this particular encoding of everything as sets we happen to take as the most basic entities.[1] This means that we can study sets also axiomatically, just as we study any other mathematical entity: as things that satisfy certain axioms. In this way we can prove theorems about sets: such theorems are the theorems of *set theory*. We shall encounter some elementary set theory theorems in what follows. Since set theory is a highly developed field of study within mathematics, there are of course many other theorems which are not discussed here: our main interest is not in set theory itself, but in its use as a mathematical modeling language, particularly in computer science.

Mathematical logic, specifically the language of *first-order logic*, allows us to define axiomatic theories, and then logically deduce theorems about such theories. Each first-order logic theory has an associated *formal language*, obtained by specifying its *constants* (for example, 0) and *function symbols* (for example, $+$ and $\cdot$ for the theory of numbers), and its *predicate symbols* (for example, a strict ordering predicate $>$). Then, out of the constants, function symbols, and variables we build *terms* (for example, $(x + 0) \cdot y$, and $(x+y)\cdot z$ are terms). By plugging terms as arguments into predicate symbols, we build the *atomic predicates* (for example, $(x + y) > 0$ is an atomic predicate). And out of the atomic predicates we build *formulas* by means of the logical connectives of conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$); and of universal ($\forall$) and existential ($\exists$) quantification, to which we also add the "there exists a unique" ($\exists!$) existential quantification variant. For example, the formula

$$(\forall x)(x > 0 \Rightarrow (x + x) > x)$$

says that for each element x strictly greater than 0, $x + x$ is strictly greater than x. This is in fact a theorem

[1] What things to take as the most basic entities is itself a matter of choice. All of mathematics can be alternatively developed in the language of category theory (another axiomatic theory); so that sets themselves then appear as another kind of entity reducible to the language of categories, namely, as objects in the *category* of sets (see, e.g., [31, 32] and [34] VI.10).

for the natural numbers. Similarly, the formula

$$(\forall x)(\forall y)(y > 0 \;\Rightarrow\; ((\exists! q)(\exists! r)((x = (y \cdot q) + r) \wedge (y > r))))$$

says that for all x and y, if $y > 0$ then there exist unique q and r such that $x = (y \cdot q) + r$ and $y > r$. This is of course also a theorem for the natural numbers, where we determine the unique numbers called the quotient q and the remainder r of dividing x by a nonzero number y by means of the division algorithm. In first-order logic it is customary to always throw in the equality predicate (=) as a built-in binary predicate in the language of formulas, in addition to the domain-specific predicates, such as >, of the given theory. This is sometimes indicated by speaking about first-order logic *with equality*.

In the *formal language of set theory* there are no function symbols and no constants, and only one domain-specific binary predicate symbol, the $\in$ symbol, read *belongs to*, or *is a member of*, or *is an element of*, which holds true of an element x and a set X, written $x \in X$, if and only if x is indeed an element of the set X. This captures the intuitive notion of belonging to a "set" or "collection" of elements in ordinary language. So, if Joe Smith is a member of a tennis club, then Joe Smith belongs to the set of members of that club. Similarly, 2, 3, and 5 are members of the set *Prime* of prime numbers, so we can write $2, 3, 5 \in \mathit{Prime}$ as an abbreviation for the logical conjunction $(2 \in \mathit{Prime}) \wedge (3 \in \mathit{Prime}) \wedge (5 \in \mathit{Prime})$. The language of *first-order formulas of set theory* has then an easy description as the set of expressions that can be formed out of a countable set of variables $x, y, z, x', y', z', \ldots$ and of smaller formulas φ, φ', etc., by means of the following BNF-like grammar:

$$x \in y \;\mid\; x = y \;\mid\; (\varphi \wedge \varphi') \;\mid\; (\varphi \vee \varphi') \;\mid\; (\varphi \Rightarrow \varphi') \;\mid\; (\varphi \Leftrightarrow \varphi') \;\mid\; \neg(\varphi) \;\mid\; (\forall x)\varphi \;\mid\; (\exists x)\varphi \;\mid\; (\exists! x)\varphi$$

where we allow some abbreviations: $\neg(x = y)$ can be abbreviated by $x \neq y$; $\neg(x \in y)$ can be abbreviated by $x \notin y$; $\neg((\exists x)\varphi)$ can be abbreviated by $(\nexists x)\varphi$ (and is logically equivalent to $(\forall x)\neg(\varphi)$); $(\forall x_1)\ldots(\forall x_n)\varphi$, respectively $(\exists x_1)\ldots(\exists x_n)\varphi$, can be abbreviated by $(\forall x_1, \ldots, x_n)\varphi$, respectively $(\exists x_1, \ldots, x_n)\varphi$; and $x_1 \in y \wedge \ldots \wedge x_n \in y$ can be abbreviated by $x_1, \ldots, x_n \in y$.

As in any other first-order language, given a formula φ we can distinguish between variables that are quantified in φ, called *bound* variables, and all other, unquantified variables, called *free* variables. For example, in the formula $(\exists x)\; x \in y$, x is bound by the $\exists$ quantifier, and y is free. More precisely, for x and y any two variables (including the case when x and y are the *same* variable):

- x and y are the only free variables in $x \in y$ and in $x = y$
- x is a free variable of $\neg(\varphi)$ iff[2] x is a free variable of φ
- x is a free variable of $\varphi \wedge \varphi'$ (resp. $\varphi \vee \varphi'$, $\varphi \Rightarrow \varphi'$, $\varphi \Leftrightarrow \varphi'$) iff x is a free variable of φ or x is a free variable of φ'
- x is neither a free variable of $(\forall x)\varphi$, nor of $(\exists x)\varphi$, nor of $(\exists! x)\varphi$; we say that x is *bound* in these quantified formulas.

For example, in the formula $(\forall x)(x = y \Rightarrow x \notin y)$ the variable x is bound, and the variable y is free, so y is the only free variable.

Set theory is then specified by its *axioms*, that is, by some formulas in the above language that are postulated as true for all sets. These are the axioms ($\varnothing$), (*Ext*), (*Sep*), (*Pair*), (*Union*), (*Pow*), (*Inf*), (*AC*), (*Rep*), and (*Found*), that will be stated and explained in the following chapters. The above set of axioms is usually denoted *ZFC* (Zermelo Fraenkel set theory with Choice). *ZFC* minus the Axiom of Choice (*AC*) is denoted *ZF*. As the axioms are introduced, we will derive some theorems that follow logically as consequences from the axioms. Other such theorems will be developed in exercises left for the reader.

The above set theory language is what is called the language of *pure set theory*, in which *all elements of a set are themselves simpler sets*. Therefore, in pure set theory quantifying over elements and quantifying over sets is exactly the same thing,[3] which is convenient. There are variants of set theory where primitive elements which are not sets (called *atoms* or *urelements*) are allowed (this is further discussed in §5.6).

[2] Here and everywhere else in these notes, "iff" is always an abbreviation for "if and only if."

[3] Of course, this would not be the same thing if we were to quantify only over the elements of a *fixed* set, say A, as in a formula such as $(\forall x \in A)\; x \neq \varnothing$. But note that, strictly speaking, such a formula does not belong to our language: it is just a notational abbreviation for the formula $(\forall x)\,((x \in A) \;\Rightarrow\; (x \neq \varnothing))$, in which x is now universally quantified over all sets.

Let us now consider the process of *logical deduction*. Any first-order logic theory is specified by the *language* $\mathcal{L}$ of its formulas (in our case, the above language of set theory formulas), and by a set Γ of *axioms*, that is, by a set Γ of formulas in the language $\mathcal{L}$, which are adopted as the axioms of the theory (in our case, Γ is the set *ZFC* of Zermelo-Fraenkel axioms). Given any such theory with axioms Γ, first-order logic provides a finite set of *logical inference rules* that allow us to derive all true theorems (and only true theorems) of the theory Γ. Using these inference rules we can construct *proofs*, which show how we can reason logically from the axioms Γ to obtain a given theorem φ by finite application of the rules. If a formula φ can be proved from the axioms Γ by such a finite number of logical steps, we use the notation $\Gamma \vdash \varphi$, read, Γ *proves* (or *entails*) φ, and call φ a *theorem* of Γ. For example, the theorems of set theory are precisely the formulas φ in the above-defined language of set theory such that $ZFC \vdash \varphi$. Similarly, if *GP* is the set of axioms of group theory, then the theorems of group theory are the formulas φ in *GP*'s language such that $GP \vdash \varphi$.

A very nice feature of the logical inference rules is that they are *entirely mechanical*, that is, they precisely specify concrete, syntax-based steps that can be carried out mechanically by a machine such as a computer program. Such computer programs are called *theorem provers* (or sometimes *proof assistants*); they can prove theorems from Γ either totally automatically, or with user guidance about what logical inference rules (or combinations of such rules) to apply to a given formula. For example, one such inference rule (a rule for conjunction introduction) may be used when we have already proved theorems $\Gamma \vdash \varphi$, and $\Gamma \vdash \psi$, to obtain the formula $\varphi \wedge \psi$ as a new theorem. Such a logical inference rule is typically written

$$\frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}$$

where Γ, φ, ψ, are *completely generic*; that is, the above rule applies to the axioms Γ of *any theory*, and to *any two proofs* $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$ of any formulas φ and ψ as theorems from Γ; it can then be used to derive the formula $\varphi \wedge \psi$ as a *new* theorem of Γ. Therefore, the collection of proofs above the vertical bar of such inference rules tells us what kinds of theorems we have *already derived*, and then what is written below the horizontal bar tells us what new theorems we can derive as *logical consequences* of the ones already derived. There are several *logical inference systems*, that is, several collections of logical inference rules for first-order logic, all of equivalent proving power (that is, all prove the same theorems, and exactly the true theorems); however, some inference systems are easier to use by humans than others. A very good discussion of such inference systems, and of first-order logic, can be found in [4].

In actual mathematical practice, proofs of theorems are *not fully formalized*; that is, an explicit construction of a proof as the result of a mechanical inference process from the axioms Γ is typically not given; instead, and *informal but rigorous* high-level description of the proof is given. This is because a detailed formal proof may involve a large amount of trivial details that are perfectly fine for a computer to take care of, but would make a standard hundred-page mathematical book run for many thousands of pages. However, the informal mathematical proofs are only correct provided that *in principle*, if the proponent of the proof is challenged, he or she could carry out a detailed *formal*, and machine verifiable proof leading to the theorem from the axioms by means of the rules of logical deduction. In these notes we will follow the standard mathematical practice of giving rigorous but informal proofs. However, in a future version of these notes some specific subtheories will be provided with fully machine-based proof assistants.

Chapter 3

The Empty Set, Extensionality, and Separation

3.1 The Empty Set

The simplest, most basic axiom, the *empty set axiom*, can be stated in plain English by saying:

There is a set that has no elements.

This can be precisely captured by the following set theory formula, which we will refer to as the ($\varnothing$) axiom:

$$(\varnothing) \qquad (\exists x)(\forall y)\ y \notin x.$$

It is very convenient to introduce an auxiliary notation for such a set, which is usually denoted by $\varnothing$. Since sets are typically written enclosing their elements inside curly brackets, thus $\{1, 2, 3\}$ to denote the set whose elements are 1, 2, and 3, a more suggestive notation for the empty set would have been $\{\}$. That is, we can think of the curly brackets as a "box" in which we store the elements, so that when we open the box $\{\}$ there is nothing in it! However, since the $\varnothing$ notation is so universally accepted, we will stick with it anyway.

3.2 Extensionality

At this point, the following doubt could arise: could there be several empty sets? If that were the case, our $\varnothing$ notation would be ambiguous. This doubt can be put to rest by a second axiom of set theory, the *axiom of extensionality*, which allows us to determine when two sets are equal. In plain English the extensionality axiom can be stated as follows:

Two sets are equal if and only if they have the same elements.

Again, this can be precisely captured by the following formula in our language, which we will refer to as the (*Ext*) axiom:

$$(Ext) \qquad (\forall x, y)((\forall z)(z \in x \Leftrightarrow z \in y) \Rightarrow x = y)$$

where it is enough to have the implication $\Rightarrow$ in the formula, instead of the equivalence $\Leftrightarrow$, because if two sets are indeed equal, then logical reasoning alone ensures that they must have the same elements, that is, we get the other implication $\Leftarrow$ for free, so that it needs not be explicitly stated in the axiom. Note that, as already mentioned, extensionality makes sure that our $\varnothing$ notation for *the* empty set is unambiguous, since there is only one such set. Indeed, suppose that we were to have two empty sets, say $\varnothing_1$ and $\varnothing_2$. Then since neither of them have any elements, we of course have the equivalence $(\forall z)(z \in \varnothing_1 \Leftrightarrow z \in \varnothing_2)$. But then (*Ext*) forces the equality $\varnothing_1 = \varnothing_2$.

The word "extensionality" comes from a conceptual distinction between a formula as a linguistic description and its "extension" as the collection of elements for which the formula is true. For example, in the

theory of the natural numbers, $x > 0$ and $x + x > x$ are different formulas, but they have the same extension, namely, the nonzero natural numbers. Extension in this sense is distinguished from "intension," as the conceptual, linguistic description. For example, $x > 0$ and $x + x > x$ are in principle different conceptual descriptions, and therefore have different "intensions." They just happen to have the same extension for the natural numbers. But they may very well have different extensions in other models. For example, if we interpret + and > on the set $\{0, 1\}$ with + interpreted as exclusive or, $1 > 0$ true, and $x > y$ false in the other three cases, then the extension of $x > 0$ is the singleton set $\{1\}$, and the extension of $x + x > x$ is the empty set. As we shall see shortly, in set theory we are able to define sets by different syntactic expressions involving logical formulas. But the extension of a set expression is the collection of its elements. The axiom of extensionality axiomatizes the obvious, intuitive truism that two set expressions having the same extension denote the *same* set.

The axiom of extensionality is intimately connected with the notion of a subset. Given two sets, A and B, we say that A is a *subset* of B, or that A is *contained* in B, or that B *contains* A, denoted $A \subseteq B$, if and only if every element of A is an element of B. We can precisely define the subset concept in our formal language by means of the defining equivalence:

$$x \subseteq y \;\Leftrightarrow\; (\forall z)(z \in x \Rightarrow z \in y).$$

Using this abbreviated notation we can then express the equivalence $(\forall z)(z \in x \Leftrightarrow z \in y)$ as the conjunction $(x \subseteq y \;\wedge\; y \subseteq x)$. This allows us to rephrase the (*Ext*) axiom as the implication:

$$(\forall x, y)((x \subseteq y \;\wedge\; y \subseteq x) \Rightarrow x = y)$$

which gives us a very useful *method* for proving that two sets are equal: we just have to prove that each is contained in the other.

We say that a subset inclusion $A \subseteq B$ is *strict* if, in addition, $A \neq B$. We then use the notation $A \subset B$. That is, we can define $x \subset y$ by the defining equivalence

$$x \subset y \;\Leftrightarrow\; (x \neq y \;\wedge\; (\forall z)(z \in x \Rightarrow z \in y)).$$

Exercise 1 *Prove that for any set A, $A \subseteq A$; and that for any three sets A, B, and C, the following implications hold:*

$$(A \subseteq B \;\wedge\; B \subseteq C) \;\Rightarrow\; A \subseteq C \qquad\qquad (A \subset B \;\wedge\; B \subset C) \;\Rightarrow\; A \subset C.$$

3.3 The Failed Attempt of Comprehension

Of course, with these two axioms alone we literally have *almost nothing!* More precisely, they only guarantee the existence of the empty set $\emptyset$, which itself has nothing in it.[1] The whole point of set theory as a language for mathematics is to have a *very expressive* language, in which any *self-consistent* mathematical entity can be defined. Clearly, we need to have other, more powerful axioms for defining sets.

One appealing idea is that if we can think of some *logical property*, then we should be able to define the set of all elements that satisfy that property. This idea was first axiomatized by Gottlob Frege at the end of the 19th century as the following *axiom of comprehension*, which in our contemporary notation can be described as follows: given any set theory formula φ whose only free variable is x, there is a set whose elements are those sets that satisfy φ. We would then denote such a set as follows: $\{x \mid \varphi\}$. In our set theory language this can be precisely captured, not by a single formula, but by a parametric family of formulas,

[1] It is like having just one box, which when we open it happens to be empty, in a world where two different boxes always contain different things (extensionality). Of course, in the physical world of physical boxes and physical objects, extensionality is always true for nonempty boxes, since two physically different nonempty boxes, must necessarily contain different physical objects. For example, two different boxes may each just contain a dollar bill, but these must be *different* dollar bills. The analogy of a set as box, and of the elements of a set as the objects contained inside such a box (where those objects might sometimes be other (unopened) boxes) can be helpful but, although approximately correct, it is not literally true and could sometimes be misleading. In some senses, the physical metaphor is *too loose*; for example, in set theory there is only one empty set, but in the physical world we can have many empty boxes. In other senses the metaphor is *too restrictive*; for example, physical extensionality for nonempty boxes means that no object can be inside two different boxes, whereas in set theory the empty set (and other sets) can belong to ("be inside of") several different sets without any problem.

called an *axiom scheme*. Specifically, for each formula φ in our language whose only free variable is x, we would add the axiom

$$(\exists! y)(\forall x)(x \in y \Leftrightarrow \varphi)$$

and would then use the notation $\{x \mid \varphi\}$ as an abbreviation for the unique y defined by the formula φ. For example, we could define in this way the *set of all sets*, let us call it the *universe* and denote it $\mathscr{U}$, as the set defined by the formula $x = x$, that is, $\mathscr{U} = \{x \mid x = x\}$. Since obviously $\mathscr{U} = \mathscr{U}$, we have $\mathscr{U} \in \mathscr{U}$. Let us call a set A *circular* iff $A \in A$. In particular, the universe $\mathscr{U}$ is a circular set.

Unfortunately for Frege, his comprehension axiom was *inconsistent*. This was politely pointed out in 1902 by Bertrand Russell in a letter to Frege (see [52], 124–125). The key observation of Russell's was that we could use Frege's comprehension axiom to define the *set of noncircular sets* as the unique set NC defined by the formula $x \notin x$. That is, $NC = \{x \mid x \notin x\}$. Russell's proof of the inconsistency of Frege's system, his "paradox," is contained in the killer question: *is NC itself noncircular*? That is, do we have $NC \in NC$? Well, this is just a matter of testing whether NC itself satisfies the *formula* defining NC, which by the comprehension axiom gives us the equivalence:

$$NC \in NC \quad \Leftrightarrow \quad NC \notin NC$$

a vicious contradiction dashing to the ground the entire system built by Frege. Frege, who had invested much effort in his own theory and can be considered, together with the Italian Giuseppe Peano and the American Charles Sanders Peirce, as one of the founders of what later came to be known as first-order logic, was devastated by this refutation of his entire logical system and never quite managed to recover from it. Russell's paradox (and similar paradoxes emerging at that time, such as the Burali-Forti paradox (see [52], 104–112; see also Theorem 25), showed that we shouldn't use set theory formulas to define other sets in the freewheeling way that the comprehension axiom allows us to do: the concept of a set whose elements are those sets that are not members of themselves is inconsistent; because if such a set belongs to itself, then it does *not* belong to itself, and *vice versa*. The problem with the comprehension axiom is its *unrestricted quantification over all sets* x satisfying the property $\varphi(x)$.

Set theory originated with Cantor (see [9] for an excellent and very readable reference), and Dedekind. After the set theory paradoxes made the "foundations problem" a burning, life or death issue, all subsequent axiomatic work in set theory has walked a tight rope, trying to find *safe* restrictions of the comprehension axiom that do not lead to contradictions, yet allow us as much flexibility as possible in defining any *self-consistent* mathematical notion. Russell proposed in 1908 his own solution, which bans sets that can be members of themselves by introducing a theory of types (see [52], 150–182). A simpler solution was given the same year by Zermelo (see [52], 199–215), and was subsequently formalized and refined by Skolem (see [52], 290–301), and Fraenkel (see [52], 284–289), leading to the so-called *Zermelo-Fraenkel set theory* (*ZFC*). *ZFC* should more properly be called Zermelo-Skolem-Fraenkel set theory and includes the already-given axioms ($\varnothing$) and (*Ext*). In *ZFC* the comprehension axiom is restricted in various ways, all of them considered safe, since no contradiction of *ZFC* has ever been found, and various *relative consistency results* have been proved, showing for various subsets of axioms $\Gamma \subset ZFC$ that if Γ is consistent (i.e., has no contradictory consequences) then *ZFC* is also consistent.

3.4 Separation

The first, most obvious restriction on the comprehension axiom is the so-called *axiom of separation*. The restriction imposed by the separation axiom consists in requiring the quantification to range, not over all sets, but over the elements of some existing set. If A is a set and φ is a set theory formula having x as its only free variable, then we can use φ to define the subset B of A whose elements are all the elements of A that satisfy the formula φ. We then describe such a subset with the notation $\{x \in A \mid \varphi\}$. For example, we can define the set $\{x \in \varnothing \mid x \notin x\}$, and this is a well-defined set (actually equal to $\varnothing$) involving no contradiction in spite of the dangerous-looking formula $x \notin x$.

Our previous discussion of extensionality using the predicates $x > 0$ and $x + x > x$ can serve to illustrate an interesting point about the definition of sets using the separation axiom. Assuming, as will be shown later, that the set of natural numbers is definable as a set $\mathbb{N}$ in set theory, that any natural number is itself a set, and that natural number addition $+$ and strict ordering on numbers $>$ can be axiomatized in set theory, we

can then define the sets $\{x \in \mathbb{N} \mid x > 0\}$ and $\{x \in \mathbb{N} \mid x + x > x\}$. Note that, although as syntactic descriptions these expressions are different, as sets, since they have the same elements, the (*Ext*) axiom forces the set equality $\{x \in \mathbb{N} \mid x > 0\} = \{x \in \mathbb{N} \mid x + x > x\}$. That is, we use a syntactic description involving the syntax of a formula to *denote* an actual set, determined exclusively by its elements. In particular, formulas φ and φ' that are *logically equivalent* (for example, $(\phi \Rightarrow \phi')$ and $(\neg(\phi) \vee \phi')$ are logically equivalent formulas) always define by separation the same subset of the given set A, that is, if φ and φ' are logically equivalent we always have the equality of sets $\{x \in A \mid \varphi\} = \{x \in A \mid \varphi'\}$.

We can describe informally the separation axiom in English by saying:

> *Given any set A and any set theory formula $\varphi(x)$ having x as its only free variable, we can define a subset of A consisting of all elements x of A such that $\varphi(x)$ is true.*

The precise formalization of the separation axiom is as an *axiom scheme* parameterized by all set theory formulas φ whose only free variable is x. For any such φ the separation axiom scheme adds the formula

$$(Sep) \quad (\forall y)(\exists! z)(\forall x)(x \in z \Leftrightarrow (x \in y \ \wedge \ \varphi)).$$

The unique set z asserted to exist for each y by the above formula is then abbreviated with the notation $\{x \in y \mid \varphi\}$. But this notation does not yet describe a concrete set, since it has the variable y as parameter. That is, we first should choose a concrete set, say A, as the interpretation of the variable y, so that the expression $\{x \in A \mid \varphi\}$ then defines a corresponding concrete set, which is a subset of A. For this reason, the separation axiom is sometimes called the *subset axiom*.

Jumping ahead a little, and assuming that we have already axiomatized the natural numbers in set theory (so that all number-theoretic notions and operations have been *reduced* to our set theory notation), we can illustrate the use of the (*Sep*) axiom by choosing as our φ the formula $(\exists y)(x = 3 \cdot y)$. Then, denoting by $\mathbb{N}$ the set of natural numbers, the set $\{x \in \mathbb{N} \mid (\exists y)\ (y \in \mathbb{N}) \ \wedge \ (x = 3 \cdot y)\}$ is the set of all *multiples* of 3.

Exercise 2 *Assuming that the set $\mathbb{N}$ of natural numbers has been fully axiomatized in set theory, and in particular that all the natural numbers* 0, 1, 2, 3, *etc., and the multiplication operation*[2] $\_ \cdot \_$ *on natural numbers have been axiomatized in this way, write a formula that, using the axiom of separation, can be used to define the set of prime numbers as a subset of $\mathbb{N}$.*

Russell's Paradox was based on the argument that the notion of a set NC of all noncircular sets is inconsistent. Does this also imply that the notion of a set $\mathscr{U}$ that is a universe, that is, a *set of all sets* is also inconsistent? Indeed it does.

Theorem 1 *There is no set $\mathscr{U}$ of all sets. That is, the formula*

$$(\nexists \mathscr{U})(\forall x)\ x \in \mathscr{U}$$

is a theorem of set theory.

Proof. We reason by contradiction. Suppose such a set $\mathscr{U}$ exists. Then we can use (*Sep*) to define the set of noncircular sets as $NC = \{x \in \mathscr{U} \mid x \notin x\}$, which immediately gives us a contradiction because of Russell's Paradox. □

[2] Here and in what follows, I will indicate where the arguments of an operation like $\cdot$ or $+$ appear by underbars, writing, e.g., $\_ \cdot \_$ or $\_ + \_$. The same convention will be followed not just for basic operations but for more general functions; for example, multiplication by 2 may be denoted $2 \cdot \_$.

Chapter 4

Pairing, Unions, Powersets, and Infinity

Although the separation axiom allows us to define many sets as subsets of other sets, since we still only have the empty set, and this has no other subsets than itself, we clearly need other axioms to get the whole set theory enterprise off the ground. The axioms of pairing, union, powerset, and infinity allow us to build many sets out of other sets, and, ultimately, *ex nihilo*, out of the empty set.

4.1 Pairing

One very reasonable idea is to consider sets *whose only element is another set*. Such sets are called *singleton sets*. That is, if we have a set A, we can "put it inside a box" with curly brackets, say, $\{A\}$, so that when we open the box there is only one thing in it, namely, A. The set A itself may be big, or small, or may even be the empty set; but this does not matter: each set can be visualized as a "closed box," so that when we open the outer box $\{A\}$ we get *only one element*, namely, the inner box A. As it turns out, with a single axiom, the axiom of *pairing* explained below, we can get two concepts for the price of one: singleton sets and (unordered) pairs of sets. That is, we can also get sets whose only elements are other sets A and B. Such sets are called (unordered) *pairs*, and are denoted $\{A, B\}$. The idea is the same as before: we now enclose A and B (each of which can be pictured as a closed box) inside the outer box $\{A, B\}$, which contains exactly *two* elements: A and B, provided $A \neq B$. What about $\{A, A\}$? That is, what happens if we try to enclose A *twice* inside the outer box? Well, this set expression still contains only *one* element, namely A, so that, by extensionality, $\{A, A\} = \{A\}$. That is, we get the notion of a singleton set as a special case of the notion of a pair. But this is all still just an intuitive, pretheoretic motivation: we need to *define* unordered pairs precisely in our language.

In plain English, the axiom of *pairing* says:

> *Given sets A and B, there is a set whose elements are exactly A and B.*

In our set-theory language this is precisely captured by the formula:

$$(\textit{Pair}) \quad (\forall x, y)(\exists! z)(\forall u)(u \in z \Leftrightarrow (u = x \ \vee \ u = y)).$$

We then adopt the notation $\{x, y\}$ to denote the unique z claimed to exist by the axiom, and call it the (unordered) *pair* whose elements are x and y. Of course, by extensionality, the order of the elements does not matter, so that $\{x, y\} = \{y, x\}$, which is why these pairs are called *unordered*. We then get the *singleton* concept as the special case of a pair of the form $\{x, x\}$, which we abbreviate to $\{x\}$.

Pairing alone, even though so simple a construction, already allows us to get many interesting sets. For example, from the empty set we can get the following, interesting sequence of sets, all of them, except $\varnothing$, singleton sets:

$$\varnothing \quad \{\varnothing\} \quad \{\{\varnothing\}\} \quad \{\{\{\varnothing\}\}\} \quad \{\{\{\{\varnothing\}\}\}\} \quad \ldots$$

That is, we enclose the empty set into more and more "outer boxes," and this gives rise to an unending sequence of *different* sets. We could actually choose these sets to represent the natural numbers in set theory, so that we could define: $0 = \varnothing$, $1 = \{\varnothing\}$, $2 = \{\{\varnothing\}\}$, $\ldots$, $n + 1 = \{n\}$, $\ldots$. In this representation we

could think of a number as a sequence of nested boxes, the last of which is empty. The number of outer boxes we need to open to reach the empty box is precisely the number n that the given singleton set in the above sequence represents. Of course, if there are no outer boxes to be opened, we do not have a singleton set but the empty set $\emptyset$, representing 0. This is a perfectly fine model of the natural numbers in set theory, due to Zermelo (see [52], 199–215). But it has the drawback that in this representation the number $n+1$ has a *single* element. As we shall see shortly, there is an alternative representation of the natural numbers, due to John von Neumann,[1] in which the natural number n is a set with exactly n elements. This is of course a more appealing representation, particularly because it will allow us, in §15, to explore a wonderful analogy (and more than an analogy: a generalization!) between computing with numbers and computing with sets.

What about *ordered* pairs? For example, in the plane we can describe a point as an ordered pair (x, y) of real numbers, corresponding to its coordinates. Can pairs of this kind be also represented in set theory? The answer is *yes*. Following an idea of Kuratowski, we can *define* an ordered pair (x, y) as a special kind of unordered pair by means of the defining equation

$$(x, y) = \{\{x\}, \{x, y\}\}.$$

The information that in the pair (x, y) x is the *first* element of the pair and y the *second* element is here encoded by the fact that when $x \neq y$ we have $\{x\} \in (x, y)$, but $\{y\} \notin (x, y)$, since $\{y\} \neq \{x\}$ and we have a proper inclusion $\{y\} \subset \{x, y\}$. Of course, when $x = y$ we have $(x, x) = \{\{x\}, \{x, x\}\} = \{\{x\}, \{x\}\} = \{\{x\}\}$. That is, the inclusion $\{y\} \subseteq \{x, y\}$ becomes an *equality* iff $x = y$, and then x is both the first and second element of the pair (x, x). For example, in the above, Zermelo representation of the natural numbers, the ordered pair $(1, 2)$ is represented by the unordered pair $\{\{\{\emptyset\}\}, \{\{\emptyset\}, \{\{\emptyset\}\}\}\}$, and the ordered pair $(1, 1)$ by the unordered pair $\{\{\{\emptyset\}\}, \{\{\emptyset\}, \{\emptyset\}\}\} = \{\{\{\emptyset\}\}, \{\{\emptyset\}\}\} = \{\{\{\emptyset\}\}\}$, which in this case happens to be a singleton set.

A key property of ordered pairs is a form of extensionality for such pairs, namely, the following

Lemma 1 *(Extensionality of Ordered Pairs). For any sets x, x', y, y', the following equivalence holds:*

$$(x, y) = (x', y') \;\Leftrightarrow\; (x = x' \;\wedge\; y = y').$$

Proof. The implication ($\Leftarrow$) is obvious. To see the implication ($\Rightarrow$) we can reason by cases. In case $x \neq y$ and $x' \neq y'$, we have $(x, y) = \{\{x\}, \{x, y\}\}$ and $(x', y') = \{\{x'\}, \{x', y'\}\}$, with the subset inclusions $\{x\} \subset \{x, y\}$ and $\{x'\} \subset \{x', y'\}$, both strict, so that neither $\{x, y\}$ nor $\{x', y'\}$ are singleton sets. By extensionality, $(x, y) = (x', y')$ means that as sets they must have the same elements. This means that the unique singleton set in (x, y), namely $\{x\}$, must coincide with the unique singleton set in (x', y'), namely $\{x'\}$, which by extensionality applied to such singleton sets forces $x = x'$. As a consequence, we must have $\{x, y\} = \{x, y'\}$, which using again extensionality, plus the assumptions that $x \neq y$ and $x \neq y'$, forces $y = y'$. The cases $x = y$ and $x' \neq y'$, or $x \neq y$ and $x' = y'$, are impossible, since in one case the ordered pair has a single element, which is a singleton set, and in the other it has two different elements. This leaves the case $x = y$ and $x' = y'$, in which case we have $(x, x) = \{\{x\}\}$, and $(x', x') = \{\{x'\}\}$. Extensionality applied twice then forces $x = x'$, as desired. □

One could reasonably wish to distinguish between the *abstract concept* of an ordered pair (x, y), and a *concrete representation* of that concept, such as the set $\{\{x\}, \{x, y\}\}$. Lemma 1 gives strong evidence that this particular choice of representation faithfully captures the abstract notion. But we could choose many other representations for ordered pairs (for two other early representations of ordered pairs due to Wiener and Hausdorff see [52] 224–227). One simple alternative representation is discussed in Exercise 3 (1). Further evidence that the above definition provides a *correct* set-theoretic representation of ordered pairs, plus a general way of freeing the abstract notion of ordered pair of any "representation bias," is given in Exercise 38, and in §6.5 after that exercise.

Exercise 3 *Prove the following results:*

1. *The alternative definition of an ordered pair as:*

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

provides a different, correct representation of ordered pairs, in the sense that it also satisfies the extensionality property stated in Lemma 1.

[1] Yes, the same genius who designed the von Neumann machine architecture! This should be an additional stimulus for computer scientists to appreciate set theory.

2. *The extensionality property of ordered pairs does* not *hold for unordered pairs. That is, show that there exists an instantiation of the variables x, y, x', y' by concrete sets such that the formula*

$$\{x, y\} = \{x', y'\} \;\Leftrightarrow\; (x = x' \;\wedge\; y = y')$$

is false of such an instantiation.

4.2 Unions

Another reasonable idea is that of *gathering together the elements of various sets into a single set*, called their *union*, that contains exactly all such elements. In its simplest version, we can just consider two sets, A and B, and define their union $A \cup B$ as the set containing all the elements in A or in B. For example, if $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, then $A \cup B = \{1, 2, 3, 4, 5\}$. One could consider giving an axiom of the form

$$(\forall x, y)(\exists! z)(\forall u)(u \in z \Leftrightarrow (u \in x \;\vee\; u \in y))$$

and then define $x \cup y$ as the unique z claimed to exist by the existential quantifier, and this would be entirely correct and perfectly adequate for *finite* unions of sets.

However, the above formula can be generalized in a very sweeping way to allow forming unions not of two, or three, or n sets, but of *any* finite or infinite collection of sets, that is, of any set of sets. The key idea for the generalization can be gleaned by looking at the union of two sets in a somewhat different way: we can first form the pair $\{A, B\}$, and then "open" the two inner boxes A and B by "dissolving" the walls of such boxes. What we get in this way is exactly $A \cup B$. For example, for $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, if we form the pair $\{A, B\} = \{\{1, 2, 3\}, \{3, 4, 5\}\}$ and then "dissolve" the walls of A and B we get: $\{1, 2, 3, 3, 4, 5\} = \{1, 2, 3, 4, 5\} = A \cup B$. But $\{A, B\}$ is just a set of sets, which happens to contain two sets. We can, more generally, consider any (finite or infinite) set of sets (and in pure set theory *any* set is always a set of sets), say $\{A_1, A_2, A_3, \ldots\}$, and then form the union of all those sets by "dissolving" the walls of the $A_1, A_2, A_3, \ldots$. In plain English, such a union of all the sets in the collection can be described by the following *union axiom*:

> *Given any collection of sets, there is a set such that an element belongs to it if and only if it belongs to some set in the collection.*

This can be precisely captured by the following set theory formula:

$$(\textit{Union}) \qquad (\forall x)(\exists! y)(\forall z)(z \in y \Leftrightarrow (\exists u)(u \in x \;\wedge\; z \in u)).$$

We introduce the notation $\bigcup x$ to denote the unique set y claimed to exist by the above formula, and call it the *union* of the collection of sets x. For example, for $X = \{\{1, 2, 3\}, \{2, 4, 5\}, \{2, 3, 5, 7, 8\}\}$ we have

$$\bigcup X = \{1, 2, 3, 4, 5, 7, 8\}.$$

Of course, when X is a pair of the form $\{A, B\}$, we abbreviate $\bigcup\{A, B\}$ by the notation $A \cup B$.

Once we have unions, we can define other boolean operations as subsets of a union, using the axiom of separation (*Sep*). For example, the *intersection* $\bigcap x$ of a set x of sets is of course the set of elements that belong to all the elements of x, provided x is *not* the empty set (if $x = \varnothing$, the intersection is not defined). We can define it using unions and (*Sep*) as the set

$$\bigcap x = \{y \in \bigcup x \mid (\forall z \in x)\; y \in z\}.$$

For example, for $X = \{\{1, 2, 3\}, \{2, 4, 5\}, \{2, 3, 5, 7, 8\}\}$, we have $\bigcap X = \{2\}$.

Note that, as with union, this is a very general operation, by which we can intersect all the sets in a possibly infinite set of sets. In the case when we intersect a pair of sets, we adopt the usual notation $\bigcap\{A, B\} = A \cap B$, and the above, very general definition specializes to the simpler, binary intersection definition

$$A \cap B = \{x \in A \cup B \mid x \in A \;\wedge\; x \in B\}.$$

Exercise 4 *Prove that* $\bigcup \emptyset = \emptyset$*, and that for for any set* x *we have the identities:* $\bigcup\{x\} = \bigcap\{x\} = x$*.*

Given two sets A and B, we say that they are *disjoint* if and only if $A \cap B = \emptyset$. For an arbitrary set of sets[2] X, the corresponding, most useful notion of disjointness is not just requiring $\bigcap X = \emptyset$, but something much stronger, namely, *pairwise disjointness*. A set of sets X is called a collection of *pairwise disjoint* sets if and only if for any $x, y \in X$, we have the implication $x \neq y \Rightarrow x \cap y = \emptyset$. In particular, *partitions* are pairwise disjoint sets of sets obeying some simple requirements.

Definition 1 *Let* X *be a collection of pairwise disjoint sets and let* $Y = \bigcup X$*. Then* X *is called a* partition *of* Y *iff either (i)* $X = Y = \emptyset$*; or (ii)* $X \neq \emptyset \wedge \emptyset \notin X$*. That is, a partition* X *of* $Y = \bigcup X$ *is either the empty collection of sets when* $Y = \emptyset$*, or a nonempty collection of pairwise disjoint nonempty sets whose union is* Y*.*

For example the set $U = \{\{1,2,3\},\{2,4,5\},\{3,5,7,8\}\}$, even though $\bigcap U = \emptyset$, is *not* a collection of pairwise disjoint sets, because $\{1,2,3\} \cap \{2,4,5\} = \{2\}$, $\{2,4,5\} \cap \{3,5,7,8\} = \{5\}$, and $\{1,2,3\} \cap \{3,5,7,8\} = \{3\}$. Instead, the set $Z = \{\{1,2,3\},\{4\},\{5,7,8\}\}$ is indeed a collection of pairwise disjoint sets, and, furthermore, it is a partition of $\bigcup Z = \{1,2,3,4,5,7,8\}$. A partition X divides the set $\bigcup X$ into pairwise disjoint pieces, just like a cake can be partitioned into pieces. For example, the above set $Z = \{\{1,2,3\},\{4\},\{5,7,8\}\}$ divides the set $\bigcup Z = \{1,2,3,4,5,7,8\}$ into three pairwise disjoint pieces.

Exercise 5 *Given a set* A *of* n *elements, let* $k = 1$ *if* $n = 0$*, and assume* $1 \leq k \leq n$ *if* $n \geq 1$*. Prove that the number of different* partitions *of* A *into* k *mutually disjoint subsets, denoted* $S(n,k)$*, satisfies the following recursive definition:* $S(0,1) = 1$*;* $S(n,n) = S(n,1) = 1$ *for* $n \geq 1$*; and for* $n > k > 1$*,* $S(n,k) = S(n\text{-}1,k\text{-}1) + (k \cdot S(n\text{-}1,k))$*. That is, you are asked to* prove *that such a recursive formula for* $S(n,k)$ *is correct for all natural numbers* n *and all* k *satisfying the already mentioned constraints.*

Exercise 6 *Jumping ahead a little, let* $\mathbb{N}$ *denote the set of all natural numbers for which we assume that multiplication* $\_ \cdot \_$ *has already been defined. For each* $n \in \mathbb{N}$*,* $n \geq 1$*, define the set* $\overset{\bullet}{n}$ *of* multiples of n *as the set* $\overset{\bullet}{n} = \{x \in \mathbb{N} \mid (\exists k)(k \in \mathbb{N} \wedge x = n \cdot k)\}$*. Then for* $1 \leq j \leq n-1$ *consider the sets* $\overset{\bullet}{n} + j = \{x \in \mathbb{N} \mid (\exists y)(y \in \overset{\bullet}{n} \wedge x = y + j)\}$*. Prove that the set of sets* $\mathbb{N}/n = \{\overset{\bullet}{n}, \overset{\bullet}{n} + 1, \ldots, \overset{\bullet}{n} + (n-1)\}$ *is pairwise disjoint, so that it provides a partition of* $\mathbb{N}$ *into* n *disjoint subsets, called the* residue classes modulo n*.*

The last exercise offers a good opportunity for introducing two more *notational conventions*. The point is that, although *in principle* everything can be reduced to our basic set theory language, involving only the $\in$ and $=$ symbols and the logical connectives and quantifiers, notational conventions allowing the use of other symbols such as $\emptyset$, $\cup$, $\cap$, etc., and abbreviating the description of sets, are enormously useful in practice. Therefore, provided a notational convention is unambiguous, we should feel free to introduce it when this abbreviates and simplifies our descriptions. The first new notational convention, called *quantification over a set*, is to abbreviate a formula of the form $(\forall y)((y \in x) \Rightarrow \varphi)$ by the formula $(\forall y \in x)\ \varphi$. Similarly, a formula of the form $(\exists y)((y \in x) \wedge \varphi)$ is abbreviated by the formula $(\exists y \in x)\ \varphi$, where in both cases we assume that x is not a free variable of φ. With this abbreviation the set $\overset{\bullet}{n} = \{x \in \mathbb{N} \mid (\exists k)(k \in \mathbb{N} \wedge x = n \cdot k)\}$ can be written in a more succinct way as $\overset{\bullet}{n} = \{x \in \mathbb{N} \mid (\exists k \in \mathbb{N})\, x = n \cdot k\}$.

The second notational convention, which can be called *separation with functional expressions*, abbreviates an application of the (*Sep*) axiom defining a set of the form $\{x \in Z \mid (\exists x_1, \ldots, x_n)(x = exp(x_1, \ldots, x_n) \wedge \varphi)\}$, where x is not a free variable of φ, by the more succinct notation $\{exp(x_1, \ldots, x_n) \in Z \mid \varphi\}$, where $exp(x_1, \ldots, x_n)$ is a functional expression which uniquely defines a set in terms of the sets $x_1, \ldots, x_n$. Using this convention, we can further abbreviate the description of the set $\overset{\bullet}{n} = \{x \in \mathbb{N} \mid (\exists k \in \mathbb{N})\, x = n \cdot k\}$ to just $\overset{\bullet}{n} = \{n \cdot k \in \mathbb{N} \mid k \in \mathbb{N}\}$. Similarly, we can simplify the description of the set $\overset{\bullet}{n} + j = \{x \in \mathbb{N} \mid (\exists y)(y \in \overset{\bullet}{n} \wedge x = y + j\}$ to just $\overset{\bullet}{n} + j = \{y + j \in \mathbb{N} \mid y \in \overset{\bullet}{n}\}$.

So far, we have seen how intersections can be obtained from unions. Using the (*Sep*) axiom, we can likewise define other *boolean operations* among sets. For example, the *set difference* $A - B$ of two sets, that is, the set whose elements are exactly those elements of A that do not belong to B, is defined using union and the (*Sep*) axiom as the set

$$A - B = \{x \in A \cup B \mid x \in A \wedge x \notin B\}.$$

[2] In pure set theory, since the elements of a set are always other sets, *all* sets are sets of sets. The terminology, "set of sets," or "collection of sets" is just suggestive, to help the reader's intuition.

Similarly, the *symmetric difference* of two sets $A \boxplus B$ can be defined by the equation

$$A \boxplus B = (A - B) \cup (B - A).$$

Exercise 7 *Prove that the binary union operation $A \cup B$ satisfies the equational axioms of: (i)* associativity*, that is, for any three sets A, B, C, we have the set equality*

$$(A \cup B) \cup C = A \cup (B \cup C)$$

(ii) commutativity*, that is, for any two sets A and B, we have the set equality*

$$A \cup B = B \cup A$$

(iii) the empty set $\varnothing$ acts as an identity element *for union, that is, for any set A, we have the equalities*

$$A \cup \varnothing = A \qquad \varnothing \cup A = A$$

and (iv) idempotency*, that is, for any set A, we have the set equality*

$$A \cup A = A.$$

Furthermore, given any two sets A and B, prove that the following equivalence always holds:

$$A \subseteq B \quad \Leftrightarrow \quad A \cup B = B.$$

Exercise 8 *Prove that the binary intersection operation $A \cap B$ satisfies the equational axioms of: (i) associativity, (ii) commutativity; and (iii) idempotency. Prove also that union and intersection satisfy the two following* distributivity *equations (of $\cap$ over $\cup$, resp., of $\cup$ over $\cap$):*

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

plus the following two absorption *equations:*

$$A \cap (A \cup C) = A \qquad A \cup (A \cap C) = A$$

plus the equation

$$A \cap \varnothing = \varnothing.$$

Furthermore, given any two sets A and B, prove that the following equivalence always holds:

$$A \subseteq B \quad \Leftrightarrow \quad A \cap B = A.$$

Exercise 9 *Prove that the symmetric difference operation $A \boxplus B$ satisfies the equational axioms of associativity and commutativity plus the axioms:*

$$A \boxplus \varnothing = A$$

$$A \boxplus A = \varnothing$$

and that, furthermore, it satisfies the following equation of distributivity *of $\cap$ over $\boxplus$:*

$$A \cap (B \boxplus C) = (A \cap B) \boxplus (A \cap C).$$

Note that, because of the associativity and commutativity of binary union, binary intersection, and symmetric difference, we can extend those operations to n sets, for any natural number $n \geq 2$, by writing $A_1 \cup \ldots \cup A_n$, $A_1 \cap \ldots \cap A_n$, and $A_1 \boxplus \ldots \boxplus A_n$, respectively, with no need for using parentheses, and where the order chosen to list the sets $A_1, \ldots, A_n$ is immaterial.

Of course, with set union, as well as with the other boolean operations we can define based on set union by separation, we can construct more sets than those we could build with pairing, separation, and the empty set axiom alone. For example, we can associate to any set A another set $s(A)$, called its *successor*, by defining $s(A)$ as the set $s(A) = A \cup \{A\}$. In particular, we can consider the sequence of sets

$$\varnothing \quad s(\varnothing) \quad s(s(\varnothing)) \quad s(s(s(\varnothing))) \quad \ldots$$

which is the sequence of von Neumann *natural numbers*. This is an alternative representation for the natural numbers within set theory, in which we define $0 = \varnothing$, and $n+1 = s(n) = n \cup \{n\}$. If we unpack this definition, the von Neumann natural number sequence looks as follows:

$$0 = \varnothing, \quad 1 = \{\varnothing\} = \{0\}, \quad 2 = \{\varnothing, \{\varnothing\}\} = \{0, 1\}, \quad 3 = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\} = \{0, 1, 2\}, \quad \ldots$$

giving us the general pattern: $n + 1 = \{0, 1, \ldots, n\}$. That is, each number is precisely the set of all the numbers before it. Two important features of this representation of numbers as sets are: (i) unlike the case for the Zermelo representation in §4.1, now the number n is a set with *exactly* n elements, which are precisely the previous numbers; and (ii) $n < m$ iff $n \in m$. These are two very good properties of the von Neumann representation, since it is very intuitive to characterize a number as a set having as many elements as that number, and to think of a bigger number as a set containing all the smaller numbers.

4.3 Powersets

Yet another, quite reasonable idea to build new sets out of previously constructed ones is to form the set of all subsets of a given set A, called its *powerset*, and denoted $\mathcal{P}(A)$. For example, given the set $3 = \{0, 1, 2\}$, its subsets are: itself, $\{0, 1, 2\}$, the empty set $\varnothing$, the singleton sets $\{0\}$, $\{1\}$, $\{2\}$, and the unordered pairs $\{0, 1\}$, $\{0, 2\}$, and $\{1, 2\}$. That is,

$$\mathcal{P}(3) = \{\varnothing, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}.$$

This gives us a total of $2^3 = 8$ subsets. The existence of a power set $\mathcal{P}(A)$ for any given set A is postulated by the *powerset axiom*, which in English can be stated thus:

Given any set, the collection of all its subsets is also a set.

This can be captured precisely in our formal set theory language by the formula

$$(Pow) \quad (\forall x)(\exists! y)(\forall z)(z \in y \Leftrightarrow z \subseteq x).$$

We then use the notation $\mathcal{P}(x)$ to denote the unique set y postulated to exist, given x, by this formula.

It is trivial to show that if $U, V \in \mathcal{P}(X)$, then $U \cup V, U \cap V, U - V, U \boxplus V \in \mathcal{P}(X)$, that is, $\mathcal{P}(X)$ is *closed under all boolean operations*. Furthermore, there is one more boolean operation not defined for sets in general, but defined for sets in $\mathcal{P}(X)$, namely, *complementation*. Given $U \in \mathcal{P}(X)$, its *complement*, denoted $\overline{U}$, is, by definition, the set $\overline{U} = X - U$. As further developed in Exercises 10 and 11, $\mathcal{P}(X)$ satisfies both the equations of a boolean algebra, and, in an alternative formulation, those of a boolean ring.

Note that this closure under boolean operations *can be extended to arbitrary unions and arbitrary intersections*. To define such arbitrary unions and intersections, we need to consider sets of sets $\mathcal{U}$ whose elements are subsets of a given set X. But what are such sets? Exactly the elements of $\mathcal{P}(\mathcal{P}(X))$. Given $\mathcal{U} \in \mathcal{P}(\mathcal{P}(X))$, its union is always a subset of X, that is, $\bigcup \mathcal{U} \subseteq X$, or, equivalently, $\bigcup \mathcal{U} \in \mathcal{P}(X)$. If $\mathcal{U} \in \mathcal{P}(\mathcal{P}(X))$ is a nonempty set of subsets of X, then we likewise have $\bigcap \mathcal{U} \subseteq X$, or, equivalently, $\bigcap \mathcal{U} \in \mathcal{P}(X)$. Recall that when $\mathcal{U} = \varnothing$, the intersection $\bigcap \mathcal{U}$ is not defined. However, we can in the context of $\mathcal{P}(X)$ extend the intersection operation also to the empty family by *fiat*, defining it as: $\bigcap \varnothing = X$. Intuitively, the more sets we intersect, the smaller the intersection:

$$\bigcap \{U\} \supseteq \bigcap \{U, V\} \supseteq \bigcap \{U, V, W\} \supseteq \ldots$$

Following this reasoning, since for any $U \subseteq X$ we have $\varnothing \subseteq \{U\}$, we should always have $\bigcap \varnothing \supseteq \bigcap \{U\} = U$. Since we know that the biggest set in $\mathcal{P}(X)$ is X itself, it is then entirely natural to define $\bigcap \varnothing = X$ as we have done.

Exercise 10 *Prove that, besides the equational laws for union and intersection already mentioned in Exercises 7 and 8, for any $U, V \in \mathcal{P}(X)$, the following additional* complement *laws hold:*

$$U \cap \overline{U} = \varnothing \qquad U \cup \overline{U} = X$$

and also the following two De Morgan's *laws:*

$$\overline{U \cup V} = \overline{U} \cap \overline{V} \qquad \overline{U \cap V} = \overline{U} \cup \overline{V}.$$

The equations in Exercises 7 and 8, plus the above equations make $\mathcal{P}(X)$ into a boolean algebra.

Exercise 11 *Prove that, besides the equations for $\boxplus$ already mentioned in Exercise 9, plus the equations of associativity, commutativity, and idempotency of $\cap$ and the equation $U \cap \emptyset = \emptyset$ in Exercise 8, for any $U \in \mathcal{P}(X)$, the following additional equational law holds:*

$$U \cap X = U.$$

These laws make $\mathcal{P}(X)$ into a boolean ring*, with $\boxplus$ as the addition operation having $\emptyset$ as its identity element, and with $\cap$ as the multiplication operation having X as its identity element.*

Prove that the operations of union and complementation on $\mathcal{P}(X)$ can be defined in terms of these, boolean ring operations as follows:

$$U \cup V = (U \cap V) \boxplus (U \boxplus V)$$
$$\overline{U} = U \boxplus X.$$

That is, instead of adopting $\cup$, $\cap$, and complementation as our basic operations on $\mathcal{P}(X)$, we may alternatively choose $\boxplus$ and $\cap$ as the basic operations.

Exercise 12 *Describe in detail the sets $\mathcal{P}(\emptyset)$, $\mathcal{P}(\mathcal{P}(\emptyset))$, and $\mathcal{P}(\mathcal{P}(\mathcal{P}(\emptyset)))$.*

Prove that if A is a finite[3] *set with n elements, then $\mathcal{P}(A)$ has 2^n elements.*

Exercise 13 *Prove that for any sets A and B, and set of sets X, we have:*

$$A \subseteq B \;\Rightarrow\; \mathcal{P}(A) \subseteq \mathcal{P}(B)$$

$$\mathcal{P}(A) \cup \mathcal{P}(B) \subseteq \mathcal{P}(A \cup B)$$
$$\mathcal{P}(A) \cap \mathcal{P}(B) = \mathcal{P}(A \cap B)$$
$$\bigcup\{\mathcal{P}(x) \in \mathcal{P}(\mathcal{P}(\bigcup X)) \mid x \in X\} \subseteq \mathcal{P}(\bigcup X)$$
$$\bigcap\{\mathcal{P}(x) \in \mathcal{P}(\mathcal{P}(\bigcup X)) \mid x \in X\} = \mathcal{P}(\bigcap X).$$

Once we have powersets, we can define many other interesting sets. For example, given sets A and B, we can define the set $A \otimes B$ of all unordered pairs $\{a, b\}$ with $a \in A$ and $b \in B$ as the set

$$A \otimes B = \{\{x, y\} \in \mathcal{P}(A \cup B) \mid (x \in A \;\wedge\; y \in B)\}.$$

Similarly, we can define the set $A \times B$ of all ordered pairs (a, b) with $a \in A$ and $b \in B$, called the *cartesian product* of A and B, as the set

$$A \times B = \{\{\{x\}, \{x, y\}\} \in \mathcal{P}(\mathcal{P}(A \cup B)) \mid (x \in A \;\wedge\; y \in B)\}.$$

Given sets $A_1, \ldots, A_n$, with $n \geq 2$, we define their cartesian product $A_1 \times \overset{n}{\ldots} \times A_n$ as the iterated binary cartesian product $A_1 \times (A_2 \times (\ldots \times (A_{n-1} \times A_n) \ldots))$; and given $x_1 \in A_1, \ldots, x_n \in A_n$, we define the *n-tuple* $(x_1, \ldots, x_n) \in A_1 \times \overset{n}{\ldots} \times A_n$ as the element $(x_1, (x_2, (\ldots, (x_{n-1}, x_n) \ldots)))$. When $A_1 = A_2 = \ldots = A_n = A$, we further abbreviate $A \times \overset{n}{\ldots} \times A$ to A^n.

Using cartesian products, we can also construct the *disjoint union* of two sets A and B. The idea of the disjoint union is to avoid any overlaps between A and B, that is, to *force* them to be disjoint before building their union. Of course, A and B may not be disjoint. But we can make them so by building "copies" of A and B that *are* disjoint. This is what the cartesian product construction allows us to do. We can form a copy of A by forming the cartesian product $A \times \{0\}$, and a disjoint copy of B by forming the cartesian product $B \times \{1\}$. These sets are respectively just like A or B, except that each element $a \in A$ has now an extra marker "0" and is represented as the ordered pair $(a, 0)$; and each $b \in B$ has now an extra marker "1" and is represented as the ordered pair $(b, 1)$. Then, by using either Lemma 1 or Exercise 15 below, it is immediate to check that $(A \times \{0\}) \cap (B \times \{1\}) = \emptyset$. We then define the *disjoint union* of A and B as the set

$$A \oplus B = (A \times \{0\}) \cup (B \times \{1\}).$$

[3] We say that a set A is *finite* iff either $A = \emptyset$, or A is a finite union of singleton sets, that is, there are singleton sets $\{a_1\}, \ldots, \{a_n\}$, such that $A = \{a_1\} \cup \ldots \cup \{a_n\}$, where by the associativity and commutativity of set union (see Exercise 7) the order and the parentheses between the different union operators are immaterial. We then use the notation $A = \{a_1, \ldots, a_n\}$ for such a set. Of course, by extensionality we remove repeated elements; for example, if $a_1 = a_2$, we would have $A = \{a_1, a_2, \ldots, a_n\} = \{a_2, \ldots, a_n\}$. The number of elements of A is of course the number of *different* elements in A. For an equivalent definition of finite set later in these notes see Definition 18 in §11.

Exercise 14 *Prove that for A, B, C, and D any sets, the following formulas hold:*

$$\begin{aligned} A \otimes B &= B \otimes A \\ A \otimes \varnothing &= \varnothing \\ A \otimes (B \cup C) &= (A \otimes B) \cup (A \otimes C) \\ (A \subseteq B \ \wedge \ C \subseteq D) &\Rightarrow A \otimes C \subseteq B \otimes D. \end{aligned}$$

Exercise 15 *Prove that for A, B, C, and D any sets, the following formulas hold:*

$$\begin{aligned} A \times \varnothing &= \varnothing \times A = \varnothing \\ A \times (B \cup C) &= (A \times B) \cup (A \times C) \\ (A \cup B) \times C &= (A \times C) \cup (B \times C) \\ (A \cap B) \times (C \cap D) &= (A \times C) \cap (B \times D) \\ A \times (B - C) &= (A \times B) - (A \times C) \\ (A - B) \times C &= (A \times C) - (B \times C) \\ (A \subseteq B \ \wedge \ C \subseteq D) &\Rightarrow A \times C \subseteq B \times D. \end{aligned}$$

Exercise 16 *Prove that if A and B are finite sets, with A having n elements and B m elements, then:*

- *$A \times B$ has $n \cdot m$ elements, and*
- *$A \oplus B$ has $n + m$ elements.*

This shows that the notations $A \times B$ and $A \oplus B$ are well-chosen to suggest multiplication and addition, since cartesian product and disjoint union generalize *to arbitrary sets the usual notions of number multiplication and addition.*

4.4 Infinity

The set theory axioms we have considered so far only allow us to build *finite* sets, like the number[4] 7 , or the powersets $\mathcal{P}(7)$ and $\mathcal{P}(\mathcal{P}(7))$, or the sets $\mathcal{P}(7) \times 7$, and $\mathcal{P}(7) \otimes 7$, and so on. It is of course very compelling to think that, if we have all the natural numbers $1, 2, 3, \ldots, n, \ldots$, as finite sets, there should exist an *infinite* set containing exactly those numbers, that is, the set of all natural numbers. Note that this set, if it exists, satisfies two interesting properties: (i) $0 = \varnothing$ belongs to it; and (ii) if x belongs to it, then $s(x) = x \cup \{x\}$ also belongs to it. We call any set satisfying conditions (i) and (ii) a *successor set*. Of course, the set of natural numbers, if it exists, is obviously a successor set; but one can construct other sets bigger than the set of natural numbers that are also successor sets.

Even though in a naive, unreflective way of doing mathematics the existence of the natural numbers would be taken for granted, in our axiomatic theory of sets it must be explicitly postulated as a new axiom, called the *axiom of infinity*, which can be stated in plain English as follows:

> *There is a successor set.*

This can be precisely formalized in our set theory language by the axiom:

$$(Inf) \qquad (\exists y)(\varnothing \in y \ \wedge \ (\forall x \in y)((x \cup \{x\}) \in y)).$$

Note that the successor set y asserted to exist by this axiom is not unique: there can be many successor sets. So this axiom does not directly define for us the natural numbers. However, it does define the natural numbers *indirectly*. To see why, first consider the following facts:

Exercise 17 *Prove that:*

- *If S and S' are successor sets, then $S \cup S'$ and $S \cap S'$ are also successor sets.*

[4] In what follows, all numbers will always be understood to be represented as sets in the von Neumann representation. Except for a brief appearance in §18.4.2, the alternative, Zermelo representation will not be further used.

- *If S is a successor set, then the set of all successor sets S' such that $S' \subseteq S$ can be precisely defined as the following subset of $\mathcal{P}(S)$:*

$$\{S' \in \mathcal{P}(S) \mid (\varnothing \in S' \ \wedge \ (\forall x \in S')((x \cup \{x\}) \in S'))\}.$$

  This set is of course nonempty (S belongs to it) and its intersection

$$\bigcap \{S' \in \mathcal{P}(S) \mid (\varnothing \in S' \ \wedge \ (\forall x \in S')((x \cup \{x\}) \in S'))\}$$

  is a successor set.

Exercise 18 *Prove that if X is a set having an element $z \in X$ such that for all $x \in X$ we have $z \subseteq x$, then $\bigcap X = z$.*

We can then use these easy facts to define the set $\mathbb{N}$ of natural numbers. Let S be a successor set, which we know it exists because of the (*Inf*) axiom. We define $\mathbb{N}$ as the intersection:

$$\mathbb{N} = \bigcap \{S' \in \mathcal{P}(S) \mid (\varnothing \in S' \ \wedge \ (\forall x \in S')((x \cup \{x\}) \in S'))\}$$

which we know is a successor set because of Exercise 17.

The key question, of course, is the *uniqueness* of this definition. Suppose we had chosen a different successor set T and had used the same construction to find its smallest successor subset. Can this intersection be *different* from the set $\mathbb{N}$ that we just defined for S? The answer is emphatically *no!* It is the *same!* Why is that? Because by Exercise 17, for any successor set T, $S \cap T$ is also a successor set. And, since $S \cap T \subseteq S$, this implies that $\mathbb{N} \subseteq (S \cap T) \subseteq T$. That is, *any successor set contains $\mathbb{N}$ as a subset*. Then, using Exercise 18, we get that for *any* successor set T

$$\mathbb{N} = \bigcap \{T' \in \mathcal{P}(T) \mid (\varnothing \in T' \ \wedge \ (\forall x \in T')((x \cup \{x\}) \in T'))\}$$

as claimed. The fact that any successor set contains $\mathbb{N}$ as a subset has the following well-known induction principle as an immediate consequence:

Theorem 2 *(Peano Induction) If $T \subseteq \mathbb{N}$ is a successor set, then $T = \mathbb{N}$.*

The above induction principle is called *Peano Induction* after Giuseppe Peano, who first formulated it in his logical axiomatization of the natural numbers.[5] It is an indispensable reasoning principle used routinely in many mathematical proofs: to prove that a property P holds for all natural numbers, we consider the subset $T \subseteq \mathbb{N}$ for which P holds; then, if we can show that $P(0)$ (that is, that $0 \in T$) and that for each $n \in \mathbb{N}$ we have the implication $P(n) \Rightarrow P(s(n))$ (that is, that $n \in T \Rightarrow s(n) \in T$), then we have shown that P holds for all $n \in \mathbb{N}$. Why? Because this means that we have proved that T is a successor set, and then by Peano Induction we must have $T = \mathbb{N}$.

Note that, although a successor set must always contain all the natural numbers, in general it could also contain other elements that are not natural numbers. The set $\mathbb{N}$ we have defined, by being the *smallest* successor set possible, contains all the natural numbers and *only* the natural numbers.

Exercise 19 *Recall that in the von Neumann natural numbers we have $n < m$ iff $n \in m$. Use Peano induction to prove that the $<$ relation is a "linear order" on $\mathbb{N}$, that is, to prove the formula*

$$(\forall n, m \in \mathbb{N})\ n < m \ \vee \ m < n \ \vee n = m.$$

(Hint: Note that the property $P(n)$ stated by the formula $(\forall m \in \mathbb{N})\ n < m \ \vee \ m < n \ \vee n = m$, defines a subset $T \subseteq \mathbb{N}$ of the natural numbers).

[5] Peano's axioms are discussed in §7.3.

Chapter 5

Case Study: A Computable Model of Hereditarily Finite Sets

Intuitively, one thinks of finite sets as *finite data structures*. However, this intuition is not quite right, since given any set A, the set $\{A\}$ is a singleton set and therefore finite; but A itself need not be finite. For example, $\{\mathbb{N}\}$ is a singleton set; but of course $\{\mathbb{N}\}$ is an *infinite data structure*, which we can depict as $\{\{0, 1, 2, 3, \ldots, n, \ldots\}\}$. What we really mean to say is that finite sets, whose elements are finite sets, whose elements are finite sets, and so on, recursively, until in the end we reach the empty set, are finite data structures. Such sets are called *hereditarily finite sets*, or *HF-sets* for brevity. HF-sets are interesting, because, as we shall see, they satisfy all the set theory axioms we have encountered so far, *except the axiom of infinity*, since $\mathbb{N}$ is not a finite set.

HF-sets seem to be not only finite data structures, but *computable* ones, so that we can represent them in a computer, can effectively decide when two such sets are equal, and can write computer programs that manipulate them to perform many of the usual set-theoretic operations, like union, intersection, powerset, and so on. Is this true? Well, the best way to settle the matter is to *write a program* that conclusively shows that HF-sets are computable. Such a program should include a *precise definition* of HF-sets (which so far we have only vaguely described) as the appropriate data set which the program's operations manipulate. All this we can do in almost any programming language, but with an important *caveat*: if we use a conventional language like C, C++, or Java, there will inevitably be a *big gap* between the data representations for HF-sets in that language, and the mathematical descriptions of HF-sets. Similarly, there will also be a big gap between the programs implementing specific set-theoretic operations and their mathematical description in set theory. This means that the task of *proving* that a computable model of HF-sets programmed in a conventional language is *correct* becomes overwhelming. Indeed, to do this at all we need a *mathematical model* of the *semantics* of C, C++, Java, or whatever other language we have chosen. This is because only thus can we have a corresponding *mathematical model* of both the data representations and the functions defined by the program implementing the set-theoretic operations. And obviously only thus can we then have a *mathematical proof* that our computer implementation of HF-sets is correct. But for realistic conventional languages such semantic models are very large and quite baroque.

In summary, the problem is that conventional programming languages are *not* mathematical languages, but complex and baroque engineering artifacts. Although they *can* be modeled mathematically –which is one of the tasks for which these notes try to prepare the reader– such models are quite large and complicated. This means that in practice it would be unfeasible within the scope of these notes to really prove that, say, a Java program correctly implements HF-sets. Is there any way out of this quagmire? Why, of course, just use a programming language that is also a *mathematical* language: in this way, we never leave the world of mathematics and the proofs become enormously easier. Are there programming languages like that? Of course! They are called *declarative* programming languages. They are typically based on some relatively simple logic so that:

1. a *program* is exactly a *theory* (i.e., a set of symbols and a set of axioms) in such a logic; and

2. *computation* is *logical deduction* with the axioms of the given theory.

In these notes I wil use the Maude language [10] for defining a computable model of HF-sets and proving it correct, and also for other case studies and in exercises. Maude's *functional modules* are exactly theories in a typed equational logic; and Maude computation with a functional module is exactly an efficient form of equational deduction that uses the equations in the module from left to right as simplification rules. Three features make Maude particularly well suited for modeling HF-sets. Firstly, by being a typed language with *sorts and subsorts*, the data set of HF-sets can be exactly characterized by a sort. Secondly, by supporting *structural axioms* such as associativity and/or commutativity, the fact that, say, the sets $\{a, b, c\}$ and $\{c, b, a\}$ are the *same* set is automatically obvious. Thirdly, by supporting *user-definable syntax*, the program's syntax and the set-theoretic notation can be kept quite close, greatly enhancing readability.

5.1 HF-Sets in Maude

The following Maude functional module, named `HF-SETS`, defines HF-sets both mathematically and computationally.

```
fmod HF-SETS is                    protecting BOOL-OPS .
  sorts Magma Set .                subsort Set < Magma .
  op _,_ : Magma Magma -> Magma [ctor assoc comm] .
  op {_} : Magma -> Set [ctor] .
  op 0 : -> Set [ctor] .           *** empty set

  vars M M' : Magma .               vars S S' T : Set .

  eq [1]: M , M = M .              *** idempotency

  op _in_ : Magma Set -> Bool .    *** generalized set membership
  eq [2]: M in 0 = false .
  eq [3]: S in {S'} = S ~ S' .
  eq [4]: S in {S',M} = (S ~ S') or (S in {M}) .
  eq [5]: S, M in {M'} = (S in {M'}) and (M in {M'}) .

  op _~_ : Set Set -> Bool .                *** set equality
  eq [6]: S ~ S' = (S <= S') and (S' <= S) .

  op _<=_ : Set Set -> Bool .               *** set containment
  eq [7]: 0 <= S = true .
  eq [8]: {M} <= S = M in S .

  op _U_ : Set Set -> Set [assoc comm] .    *** union
  eq [9]: S U 0 = S .
  eq [10]: {M} U {M'} = {M,M'} .

  op P : Set -> Set .                       *** powerset
  eq [11]: P(0) = {0} .
  eq [12]: P({S}) = {0,{S}} .
  eq [13]: P({S,M}) = P({M}) U augment(P({M}),S) .

  op augment : Set Set -> Set .
  eq [14]: augment(0,T) = 0 .
  eq [15]: augment({S},T) = {{T} U S} .
  eq [16]: augment({M,M'},T) = augment({M},T) U augment({M'},T) .
endfm
```

Let me explain this module in detail. Any Maude functional module is introduced with the keyord `fmod` and ended with the keyword `endfm`. This is followed by its name, in this case `HF-SETS`, and possibly by a list of imported modules. In this case, the `protecting BOOL-OPS` declaration imports the module `BOOL-OPS`, which in particular contains the constants `true` and `false` and the operators `_and_` and `_or_` used in `HF-SETS`. A theory in a logic contains a set of symbols and a set of axioms. Of course, all the symbols and axioms of `BOOL-OPS` are included, plus the ones explicitly declared in `HF-SETS`. In Maude the symbols are *typed*, with the types called *sorts*. In addition to the `Bool` sort already declared in `BOOL-OPS`, two new sorts, namely, `Set` and `Magma` are declared in the `sorts` declaration. The intention is that an HF-set of the form $\{a, b, c\}$ will have sort `Set`, whereas the data structure a, b, c obtained by removing the outer curly brackets will have sort `Magma`. Furthermore, the `subsort` declaration states that any `Set` is also a `Magma`, namely, a magma with

a single element. That is, not only is, say, $\emptyset, \{\emptyset\}, \{\{\emptyset\}\}$ a magma, but $\emptyset, \{\{\emptyset\}\}$ and $\{\emptyset\}, \{\{\emptyset\}\}$ and $\emptyset, \{\emptyset\}$ and $\emptyset$ and $\{\emptyset\}$ and $\{\{\emptyset\}\}$ are also magmas, and the last three are also sets. Once the sorts and subsorts have been declared, we are ready to define the *function symbols* of our theory. Each such symbol, say f, is declared with syntax `op` f : $s_1 \ldots s_n$ `->` s `.`, with $s_1 \ldots s_n$, the sorts for the arguments and s the result sort. Furthermore, the symbol f can be declared with user-definable syntax as either a single identifier in *prefix* syntax, as done for the `P` and `augment` function symbols, so that then an expression with top symbol f is of the form $f(t_1, \ldots, t_n)$ as in, e.g., `P({S,M})` and `augment({S},T)`, or with *mix-fix* syntax, e.g., `_and_`, `_or_` and `_U_`, with as many underbars as argument sorts, so that then we obtain expressions such as `true and false`, `true or false`, and `S U 0`. Note that *constants* are viewed as the special case of function symbols with zero argument sorts. For example the constant `0` is used as typewriter notation for the empty set $\emptyset$ and is declared to have sort `Set`. Maude supports the distinction between *constructor* function symbols, used to build up data structures, and *defined* function symbols, which take such data structures as arguments and return other data structures as their result. Any function symbol with the `ctor` attribute is declared to be a constructor. In our case, the only data constructors are `0`, `{_}`, and `_,_` (as well as `true` and `false` for the `Bool` sort). All the other function symbols are defined functions. In addition to its `ctor` attribute, the symbol `_,_` is also declared with the associativity, `assoc`, and commutativity, `comm`, structural axioms. This means that `_,_` satisfies the structural axioms `(M , M') , M'' = M , (M' , M'')` and `M , M' = M' , M`, so that Maude can use such axioms when computing with expressions involving `_,_`. Note that this means that *we do not need parentheses* to disambiguate between, say, $(a, b), c$ and $a, (b, c)$, so we can just write a, b, c. It also means that, *order is immaterial*, so that, e.g., $a, b, c = c, b, a$. Note that the union operator `_U_` is also declared to be associative and commutative.

Beside the structural equational axioms, such as `assoc` and `comm`, and all the axioms in `BOOL-OPS`, what other axioms are declared in `HF-SETS`? The module declares 16 equations with the `eq` keyword. The variables used in such equations as well as their respective sorts are listed in two `vars` declarations. Such equations can be optionally labeled, which is done here for ease of reference. Equation `[1]` is an idempotency equation for the constructor `_,_`. This means that, for example, we have the equality of magmas $a, a, b, b, b, c = a, b, c$ and therefore the equality of sets $\{a, a, b, b, b, c\} = \{a, b, c\}$, as we would expect. This idempotency equation as well as the `assoc` and `comm` axioms for `_,_` say all that needs to be said about the equalities that hold for the data structures of HF-sets and magmas. The remaining equations `[2]`–`[16]` define different functions taking sets (and possibly magmas) as arguments. Equations `[2]`–`[5]` define the membership predicate $\in$, here denoted `_in_`. We allow the more general notation $a_1, \ldots, a_n \in A$ introduced in §2 as an abbreviation for $a_1 \in A \wedge \ldots \wedge a_n \in A$. In fact, equation `[5]` defines away this abbreviation in terms of the standard membership relation $a \in A$. Equations `[7]` and `[8]` define the subset relation $\subseteq$, here denoted `_<=_`. The set equality predicate $=$, here denoted `_~_`, is defined away in the expected manner as mutual set containment. Equations `[9]` and `[10]` define the set union operator $\cup$, here denoted `_U_`. And equations `[11]`–`[13]` define the powerset operator $\mathcal{P}$, here denoted `P`. Finally, equations `[14]`–`[16]` define the auxiliary function `augment`, used in defining `P`, which adds an extra element to all the sets that are themselves elements of a given set.

Therefore, the claim that a Maude functional module is indeed a *theory* consisting of symbols and axioms is now fully explained for our `HF-SETS` module. The axioms, all of them equations, are of course axioms in first-order logic, with the slight qualification of using a *typed* version of first-order logic, whereas the standard version is untyped. How about *computing* in such a module? According to the general definition of a declarative language given above, this should of course correspond to *proving* a formula using the axioms. Since our axioms are equations, computations should be proofs in our typed equational logic, which is a sublogic of (typed) full first-order logic. By restricting to a simpler logic, and even to simpler kinds of proofs in that sublogic, this kind of logical deduction can be implemented very efficiently, so that declarative programming, besides being vastly superior intellectually, can compete in efficiency with conventional programming. For example, Maude can perform millions of logical deduction steps per second on conventional machines.

What exactly are the equational proofs performed by Maude? I give a full answer to this question in §5.2; for the moment let me just explain the main ideas informally and illustrate them with example computations. Equational logic is the logic of *replacing equals by equals* that we all used in grammar school when we first computed with algebraic expressions. For example, if a commutativity equation $x + y = y + x$ is one of our axioms, then we can prove that $a * (b + (0 * c)) = a * ((0 * c) + b)$ by applying our

equation to the subexpression $(b + (0 * c))$. Of course, replacement of equals by equals can be performed in either direction, from left to right, or from right to left, which makes it undirected and inefficient. In practice, however, many algebraic equations have a natural orientation from left to right as *simplification rules*. For example, the equations $x + 0 = x$, $x * 0 = 0$, and $x + -x = 0$ are typically applied from left to right to obtain simpler expressions. Maude applies all the equations declared with the `eq` keyword from left to right in exactly this way to prove that the input term we give Maude for reduction is equal to a constructor term which is the result of simplifying the input term. But what about the structural axioms such as `assoc` and `comm`? Obviously, an equation such as $x + y = y + x$ *cannot* be used as a simplification rule, because it will lead to looping, nonterminating deductions. What we can do, however, is to use such structural axioms to make each step of simplification more powerful. How so? By allowing a simplification equation to be applied not just to a given term, but to any term equivalent to it *modulo* the structural axioms, such as `assoc` and/or `comm`. For example, if $x * y = y * x$ has been specified by a `comm` declaration for $*$, although we cannot directly apply the simplifying equation $x * 0 = 0$ to the term $a * (b + (0 * c))$, we can apply it to its equivalent term $a * (b + (c * 0))$ (modulo the equation $x * y = y * x$) to get the simpler term $a * (b + c)$. Similarly, if + is declared with the `assoc` and `comm` axioms, we can apply the simplifying equation $x + -x = 0$ to the expression $a + (b + -a)$ to obtain the expression $b + 0$, because we can prove $a + (b + -a) = b + (a + -a)$ using the associativity and commutativity equations. This is exactly what Maude does, using a very efficient built-in way to compare equality of terms modulo axioms such as `assoc` and/or `comm` each time that a simplification equation is applied. Let me illustrate this idea with a simple example. Consider the HF-set expression `{0,{0,0},0,{0}}`. Since this is an expression involving only constructors, only equation `[1]` could possibly be applied to simplify it. We can ask Maude to simplify it by giving the `red` command. Also, if we want to see the *equational proof* that Maude builds, we can give first the `trace` command.

```
Maude> set trace on .
Maude> red {0,{0,0},0,{0}} .
reduce in HF-SETS : {0,{0,0},0,{0}} .
*********** equation
eq M,M = M [label 1] .
M --> 0
0,0
--->
0
*********** equation
eq M,M = M [label 1] .
M --> {0}
0,{0},{0}
--->
0,{0}
*********** equation
eq M,M = M [label 1] .
M --> 0
0,0,{0}
--->
{0},0
result Set: {0,{0}}
```

Let me explain this trace in detail. There have been three applications of equation `[1]` before reaching the fully simplified set expression `{0,{0}}`. The first and third applications instantiated the variable `M` of equation `[1]` to `0`, whereas the second instantiated `M` to `{0}`. Let us use $u =_{AC} v$ for the structural equality obtained by repeated application of the `assoc` and `comm` axioms, and $u \rightarrow_{[1]} v$ for a left-to-right simplification step with equation `[1]`. The equational proof built by Maude is then,

$$\texttt{\{0,\{0,0\},0,\{0\}\}} \rightarrow_{[1]} \texttt{\{0,\{0\},0,\{0\}\}} =_{AC} \texttt{\{0,\{0\},\{0\},0\}} \rightarrow_{[1]} \texttt{\{0,\{0\},0\}} =_{AC} \texttt{\{0,0,\{0\}\}} \rightarrow_{[1]} \texttt{\{0,\{0\}\}}$$

Let us finish this section by showing some sample reductions for the different defined function symbols.

```
Maude> red 0 in {0,{0,0},0,{0}} .
result Bool: true

Maude> red {{0}} in {0,{0,0},0,{0}} .
result Bool: false
```

```
Maude> red {0,{0,0},0,{0}} ~ {0,{0}} .
result Bool: true

Maude> red {0,{0,0},0,{0}} ~ {{0}} .
result Bool: false

Maude> red {{0,0},{0}} <= {0,{0}} .
result Bool: true

Maude> red {0,{0}} <= {{{0}},{0}} .
result Bool: false

Maude> red {{0,{0}},{0}} U  {0,{0},{{0}}} .
result Set: {0,{0},{{0}},{0,{0}}}

Maude> red P({0}) .
result Set: {0,{0}}

Maude> red P({0,{0}}) .
result Set: {0,{0},{{0}},{0,{0}}}

Maude> red P({0,{0},{{0}},{0,{0}}}) .
result Set: {0,{0},{{0}},{{{0}}},{{0,{0}}},{0,{0}},{0,{{0}}},{0,{0,{0}}},{{0},{
    {0}}},{{0},{0,{0}}},{{{0}},{0,{0}}},{0,{0},{{0}}},{0,{0},{0,{0}}},{0,{{0}},
    {0,{0}}},{{0},{{0}},{0,{0}}},{0,{0},{{0}},{0,{0}}}}

Maude> red augment({0,{0},{{0}},{0,{0}}},{{{0}}}) .
result Set: {{{{{0}}}},{0,{{{0}}}},{{0},{{{0}}}},{0,{0},{{{0}}}}}
```

Of course, although we have an equational theory that it is claimed to correctly *model* HF-sets, we do not yet have a mathematical statement and proof of this claim: this matter will be taken up in §5.4. But before, we have to take a closer look, in §5.2 and §5.3, at the equational proofs built by Maude.

Exercise 20 *Implement HF-sets in your favorite conventional language, e.g., C, C++, or Java. Implement also HF-sets in your favorite declarative language, e.g., Haskell or Prolog. Compare the pros and cons of your different implementations, e.g., in terms of conciseness, understandability, and efficiency. Note that we are after a* model *of HF-sets, so tests for what counts as a HF-set and for when two different data representations are equal are essential and must be available in your implementations. Think about what it would be like in each case to give a mathematical proof that your implementation correctly models HF-sets.*

5.2 Terms, Equations, and Term Rewriting

`HF-SETS` is just *one* Maude functional module. But what are Maude functional modules like in general? And what kind of equational deduction is performed by a Maude reduction? The answer to the first question is easy: ignoring module importation issues (we can always assume that all the imported symbols and axioms are explicitly added inside a single module), and declarations such as `ctor` which just add extra information, a Maude functional module is just a *typed equational theory* with typed function symbols, say, Σ, and with two sets of equational axioms, B and E, where B is a set of *structural axioms* such us associativity, and/or commutativity, and/or identity axioms for some operators in Σ, which in Maude are declared with the `assoc`, `comm`, and `id:` keywords, and E is a set of equations[1] to be used from left to right as *simplification rules*. There are also some *executability assumptions* for E which I will discuss in §5.3. Such an equational theory can be succinctly described as a pair $(\Sigma, E \cup B)$, so that a Maude functional module is just a typewriter notation for specifying it as the textual declaration fmod $(\Sigma, E \cup B)$ endfm. To answer the question about what kind of equational deduction is performed by a Maude reduction, we need to precisely describe what *proofs* are for any such theory $(\Sigma, E \cup B)$. But for this we need to say what *terms* (i.e., expressions) are for the typed syntax Σ. And for this, in turn, we need to say more precisely what the typed syntax Σ, called a *signature*, is. So let us start with signatures.

[1] As further explained in §5.5, the equations in E can be *conditional*, and E may also contain *membership axioms*, specifying what sorts a term may have under certain conditions.

Definition 2 *An* order-sorted signature *is a pair* $\Sigma = ((S, <), F)$*, with* $(S, <)$ *a partially ordered set*[2] *whose elements are called* sorts*. If* $s < s'$ *we say that sort* s *is a* subsort *of sort* s'*.* F*, called the set of* function symbols*, is then a set of triples of the form* $(f, s_1 \ldots s_n, s)$*, which we display as* $f : s_1 \ldots s_n \longrightarrow s$*, where* $s_1 \ldots s_n$ *is a sequence of sorts in* S*, called the* argument sorts*, and* $s \in S$ *is called the* result sort*. When* $n = 0$*, we call* (f, nil, s)*, with nil the empty sequence, a* constant*.*

In Maude the sorts in S are specified with the `sorts` declaration, and subsort relations with `subsort` declarations. Function symbols are of course specified with `op` declarations. Note that, for simplicity, we are assuming a *prefix syntax* for each function symbol f. In Maude it is always possible to write any term in prefix form. For example, `true or false` and `_or_(true,false)` are treated as equivalent terms by the Maude parser.

Given a signature Σ, we are now ready to define its terms, i.e., the parseable expressions for the symbols Σ. Note that constants behave as terminal symbols in a grammar, sorts behave as nonterminal symbols, and function symbol declarations behave as grammar rules. I will use the notation $t : s$ to indicate that a term t has sort s.

Definition 3 *Given an order-sorted signature*[3] $\Sigma = ((S, <), F)$*, a* Σ-term t *of sort* $s \in S$*, denoted* $t : s$*, is a syntactic expression that can be obtained by finite application of the following term formation rules:*

1. *For each* $f : s_1 \ldots s_n \longrightarrow s$ *in* F*, if* $t_1 : s_1, \ldots, t_n : s_n$*, then* $f(t_1, \ldots, t_n) : s$*. In particular, if* $a : nil \longrightarrow s$ *is in* F*, then* $a : s$*.*

2. *If* $t : s$ *and* $s < s'$*, then* $t : s'$*.*

We denote by T_Σ *the set of all* Σ*-terms of any sort, and by* $T_{\Sigma,s}$ *the set of* Σ*-terms of sort* s*. Note that, by rule (2), if* $s < s'$*, then we necessarily have* $T_{\Sigma,s} \subseteq T_{\Sigma,s'}$*.*

Note that the Σ-terms defined above do not contain any variables: they only contain constants and other function symbols. They are sometimes called *ground* Σ-terms to make this fact explicit. However, the case of terms with variables, like $(f(x, g(a, y))$, where a is a constant and x, y are variables, is just a special case of the above definition. Given a signature $\Sigma = ((S, <), F)$ and a (finite or infinite) set of typed variables, say, $X = \{x_1 : s_1, \ldots, x_n : s_n, \ldots\}$ (where we assume that the *names* $x_1, \ldots, x_n, \ldots$ are all *different*, and different from the constants in Σ), we can just extend Σ to a bigger signature $\Sigma(X) = ((S, <), F(X))$ by adding the variables in X as extra constans, i.e., $F(X) = F \cup \{(x_1, nil, s_1), \ldots, (x_n, nil, s_n), \ldots\}$. Then a Σ-*term with variables* in X is, by definition, just a (ground) $\Sigma(X)$-term.

In a Σ-term $f(t_1, \ldots, t_n)$, the $t_1, \ldots, t_n$ are called its *immediate subterms*. A term u is called a *subterm* of $f(t_1, \ldots, t_n)$ if either: (i) $u = f(t_1, \ldots, t_n)$, or (ii) u is an immediate subterm of $f(t_1, \ldots, t_n)$, or (iii) there is a finite sequence of terms $f(t_1, \ldots, t_n)$, t_i, $v_1, \ldots, v_k$, u (where $k \geq 0$) such that each next term in the sequence is an immediate subterm of the preceding one. Note that a term may contain different occurrences of the same subterm. For example, the subterm $g(a)$ appears twice in the term $f(b, h(g(a)), g(a))$. One way to make clear *where* a subterm is located is to replace such a subterm by a single *hole*, that is by a fresh blank constant [] indicating where the subterm was before we removed it. For example, we can indicate the two places where $g(a)$ occurs in $f(b, h(g(a)), g(a))$ by $f(b, h([]), g(a))$ and $f(b, h(g(a)), [])$. A term with a *single occurrence of a hole* is called a *context*. We write $C[]$ to denote such a term. Given a context $C[]$ and a term u, we can obtain a new term, denoted $C[u]$, by replacing the hole [] by the term u. For example, if $C[] = f(b, h([]), g(a))$ and $u = k(b, y)$, then $C[u] = f(b, h(k(b, y)), g(a))$. Of course, if $C[]$ is the context obtained from a term t by placing a hole [] were subterm u occurred, then we have the term identity $t = C[u]$, that is, we can always *decompose* a term into a context and a chosen subterm. For example, we have, among others, the following decompositions of our term $f(b, h(g(a)), g(a))$:

$$f(b, h(g(a)), g(a)) = f(b, h([g(a)]), g(a)) = f(b, h(g(a)), [g(a)]) = f(b, [h(g(a))], g(a)) = [f(b, h(g(a)), g(a))]$$

[2]Partially ordered sets will be studied in detail in §9.4; for the moment all we need to know is that $<$ is a subset of $S \times S$, with $(s, s') \in <$ denoted $s < s'$. $(S, <)$ being a partial order just means that we cannot have $x < x$ (irreflexivity) and that $x < y$ and $y < z$ implies $x < z$ (transitivity).

[3]From now on I will assume that the signature Σ is *preregular*, which is a syntactic condition ensuring that for each term t there is a *smallest* sort s such that $t : s$. Maude automatically checks this condition, and even the stronger condition of *preregularity modulo* associativity and/or commutativity and/or identity axioms (see [10], 22.2.5).

where the last is a decomposition where the top part is the "empty context" []. This is very useful, since such decompositions indicate *where* in a term we either have changed one subterm by another, or could perfom such a change.

Note that in an order-sorted setting we can write terms that correspond to error expressions, since, though still *meaningful*, there is no sort that can be assigned to them. For example, the term `{false}` is *meaningless*, since Boolean elements and sets are totally unrelated kinds of values in the `HF-SETS` module. Instead, the term `P(0,{0})` is an *error term*, since no sort can be assigned to it, given that `P`'s argument is of sort `Magma`, but `P` expects arguments of sort `Set`. However, the term `P(0,{0})` is still meaningful, since, although there is no way to evaluate the particular term `P(0,{0})` to a well-formed term, we can think of similar error terms that *can* be evaluated to a well-foremd term. For example, the error term `P({0},{0})` can be evaluated to the well-formed term `P({0})` by using the idempotency equation. This suggests the idea of giving such erroneous but meaningful terms the benefit of the doubt and to wait until they are fully evalauted to determined whether they are either harmless ill-formed terms like `P({0},{0})` which denote a well-formed value, or truly erroneous expressions like `P(0,{0})`. This is exactly what Maude does by automatically adding a *kind* above each connectd component of the poset of sorts for the given module. If we picture $(S, <)$ as a graph, with $s < s'$ wiewed as an edge $s \to s'$, its is a directed acyclic graph whose connected components are the different clusters of nodes connected by the edges of the graph. For example, in the `HF-SETS` module, there are two connected components: one containing the sorts `Set` and `Magma`, and another containing only the sort `Bool`. A detailed definition of a poset's connected components is given in Exercise 75. Maude automatically adds an extra "kind" sort on top of each connected component, which is denoted $[s]$ for s any sort in that component. For `HF-SETS` Maude adds the kinds `[Magma]` and `[Bool]` and subsort relations `Magma < [Magma]` and `Bool < [Bool]`. The term `{false}` still remain meaningless, but the terms `P(0,{0})` and `P({0},{0})` are now well-formed terms of kind `[Magma]`. In short, Maude automatically completes the user-given order-sorted signature to a *kind-complete* one in the following sense.

Definition 4 *An order-sorted signature $\Sigma = ((S, <), F)$ is called* kind-complete *if each connected component of the poset $(S, <)$ has a top element, denoted*[4] *$[s]$, for s any other sort in that component, so that $s < [s]$. Furthermore, for any $f : s_1 \ldots s_n \longrightarrow s$ in F, $(n \geq 1)$, there is also an $f : [s_1] \ldots [s_n] \longrightarrow [s]$ in F.*

Given an order-sorted signature Σ, a Σ-*equation* is an atomic formula $t = t'$, where $t, t' \in T_{\Sigma(X)}$, with $X = \{x_1 : s_1, \ldots, x_n : s_n\}$ the variables appearing as subterms in either t or t'. An equation $t = t'$ must always be *well typed*. For example, in our `HF-SETS` module, the "equation" `0 = true` is utter nonsense, since `0` has sort `Set`, but `true` has sort `Bool`. For an equation $t = t'$ to be well typed we must require that there are sorts $s, s' \in S$, *both in an identical connected component* of the poset $(S, <)$, such that $t : s$ and $t' : s'$. For example, in `HF-SETS` the equation `S , S = S` is well typed, because `S , S` has sort `Magma`, `S` has sort `Set`, and the sorts `Set` and `Magma` are related in the subset order and therefore belong to the same connected component. In an equational theory (Σ, E) all equations $t = t' \in E$ are implicitly assumed to be *universally quantified* as $(\forall x_1 : s_1, \ldots, x_n : s_n)\ t = t'$, with $vars(t = t') = \{x_1 : s_1, \ldots, x_n : s_n\}$ the set of variables appearing as subterms in t or t'.

Let us go back to the equational proof step $a * (b + (0 * c)) = a * ((0 * c) + b)$, which applied the equation $x + y = y + x$ to the subterm $(b + (0 * c))$. Note that this involved *instantiating* the variable x to the term b, and the variable y to the term $(0 * c)$. Such an instantiation is called a *substitution*, and can be precisely described as the set $\{(x, b), (y, (0 * c))\}$.

Definition 5 *(Substitutions). Let Σ be an order-sorted signature, and let X, Y be sets of variables with X finite, say, $X = \{x_1 : s_1, \ldots, x_n : s_n\}$. Then a* substitution *$\theta$, denoted $\theta : X \longrightarrow T_{\Sigma(Y)}$, mapping the variables of X to Σ-terms with variables in Y, is a set of the form $\theta = \{(x_1, t_1), \ldots, (x_n, t_n)\}$, with $t_1, \ldots, t_n \in T_{\Sigma(Y)}$, and with $t_i : s_i$, $1 \leq i \leq n$. Given a term $t \in T_{\Sigma(X)}$, its* instantiation *by θ, denoted $\theta(t)$, is defined inductively by:*

1. *$\theta(x_i) = t_i$, $1 \leq i \leq n$.*
2. *$\theta(a) = a$ for each constant $a : nil \longrightarrow s$ in Σ.*
3. *$\theta(f(u_1, \ldots, u_k)) = f(\theta(u_1), \ldots, \theta(u_k))$.*

[4] By convention, if s, s' are two non-top sorts in a connected component of $(S, <)$, then $[s]$ and $[s']$ both denote the unique top sort $[s] = [s']$ in that component.

For example, for $\theta = \{(x, b), (y, (0 * c))\}$, we have $\theta(x + y) = b + (0 * c)$, and $\theta(y + x) = (0 * c) + b$. We are now ready to characterize equational simplification proofs and equational proofs more precisely.

Definition 6 *(Rewrite Proofs and Equational Proofs).*[5] *Let Σ be a kind-complete signature, E a set of Σ-equations, and Y a set of typed variables with infinitely many variables for each sort $s \in S$. Then an E-*rewrite step *(called also an E-*simplification step*) is a pair $(u, v) \in T^2_{\Sigma(Y)}$, denoted $u \to_E v$, such that there is a term decomposition $u = C[w]$, an equation $t = t' \in E$, and a substitution $\theta : vars(t = t') \longrightarrow T_{\Sigma(Y)}$ such that $w = \theta(t)$, and $v = C[\theta(t')]$. Similarly, an E-*equality step *for terms with variables in Y, denoted $u \leftrightarrow_E v$, is, by definition, an $(E \cup E^{-1})$-rewrite step, where $E^{-1} = \{t' = t \mid t = t' \in E\}$. Rewrite proofs (resp., equational proofs) are then chains of rewite (resp., equational) steps:*

- *An E-*rewrite proof *is either (i) a pair $(u, u) \in T^2_{\Sigma(Y)}$, denoted $u \to^*_E u$, or (ii) a pair $(u, v) \in T^2_{\Sigma(Y)}$, denoted $u \to^*_E v$, such that there is a chain of E-rewrite steps $v_0 \to_E v_1 \to_E v_2 \ldots v_{n-1} \to_E v_n$, $n \geq 1$, with $u = v_0$ and $v = v_n$.*

- *An E-*equality *proof, denoted $u =_E v$, is, by definition, an $(E \cup E^{-1})$-*rewrite proof *$u \to^*_{E \cup E^{-1}} v$.*

Recall that in Maude the rewrite proofs with equations E are performed *modulo* some axioms B, such as associativity, and/or commutativity, and/or identitiy axioms. Therefore, unless the functional module in question is such that $B = \emptyset$, rewrite proofs $u \to^*_E v$ are not sufficient. We need instead the more general notion of rewrite proofs *modulo B*.

Definition 7 *(Rewrite Proofs Modulo B). Let E and B be sets of Σ-equations, and let Y be a set of typed variables with infinitely many variables for each sort $s \in S$. Then an E-*rewrite step modulo *B is a pair $(u, v) \in T^2_{\Sigma(Y)}$, denoted $u \to_{E/B} v$, such that there are terms $u', v' \in T_{\Sigma(Y)}$ such that $u =_B u'$, $u' \to_E v'$, and $v' =_B v$, that is, we have $u =_B u' \to_E v' =_B v$. An E-*rewrite proof modulo *B is either (i) a pair $(u, u') \in T^2_{\Sigma(Y)}$, denoted $u \to^*_{E/B} u'$, such that $u =_B u'$, or (ii) a pair $(u, v) \in T^2_{\Sigma(Y)}$, denoted $u \to^*_{E/B} v$, such that there is a chain of E-rewrite steps modulo B $v_0 \to_{E/B} v_1 \to_{E/B} v_2 \ldots v_{n-1} \to_{E/B} v_n$, $n \geq 1$, with $u = v_0$ and $v = v_n$.*

We can now come back to the earlier question: what kind of equational deduction is performed by a Maude reduction? The answer is: if the functional module is of the form `fmod` $(\Sigma, E \cup B)$ `endfm`, then when we submit to Maude a term t with the `red` command, Maude builds a *terminating* proof of the form $t \to^*_{E/B} t'$ such that no more E-rewrite steps modulo B are possible from t', that is, there is no t'' such that $t' \to_{E/B} t''$. I will use the notation $t \to^!_{E/B} t'$ for such terminating proofs. Therefore, if t was a ground term (we can also submit to Maude terms with variables for reduction), and the equations E satisfy the requirements explained in §5.3, t' should be a constructor term. Note that any rewrite proof $t \to^*_{E/B} t'$, and in particular any terminating proof $t \to^!_{E/B} t'$, is *a fortiori* an equational proof $t =_{E \cup B} t'$. But rewrite proofs can be performed much more efficient than arbitrary equational proofs.

5.3 Confluence, Termination, and Sufficient Completeness

Given a Maude functional module fmod $(\Sigma, E \cup B)$ endfm, what *executabilty conditions* should be placed on its equations E so that we can compute with it and obtain appropriate answers to our queries? The most basic requirement is: (1) for each $t = t' \in E$, *any variable x occuring in t' must also occur in t*. Otherwise, $t = t'$ is ill suited for equational simplification, since we have to *guess* how to instantiate the extra variables in t', and there can be an infinity of guesses. For example, if we tried to use the equation $0 = x * 0$ from left to right to "simplify" the term $(a + b) + 0$, we could obtain $(a + b) + (0 * 0)$, or $(a + b) + (z * 0)$, or $(a + b) + ((c * d) * 0)$, and so on, depending on whether the substitution we chose was $\theta = \{(x, 0)\}$, or $\theta = \{(x, z)\}$, or $\theta = \{(x, (c * d))\}$, and so on. Instead, if we apply the equation $x + 0 = x$ to the same term, there is only *one* substitution possible to simplify $(a + b) + 0$, namely, $\theta = \{(x, (a + b)\}$. Note that, under

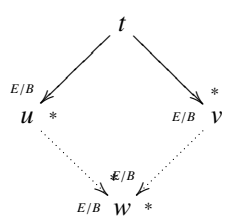
[5] To simplify the exposition I am assuming that the order-sorted signature Σ has *nonempty sorts*, that is, for each $s \in S$ we have $T_{\Sigma,s} \neq \emptyset$. If Σ has some empty sorts, a treatment with explicitly quantified equations $(\forall x_1 : s_1, \ldots, x_n : s_n)\ t = t'$ is needed. See, e.g., [41].

assumption (1), if $u \to_E v$ and $X = \{x_1 : s_1, \ldots, x_n : s_n\}$ are the variables appearing in u, then no new variables are introduced in v, that is, $v \in T_{\Sigma(X)}$.

A second important requirement is: (2) *sort-decreasingness*, that is, for each equation $t = t' \in E$, sort $s \in S$, and substitution θ we must have the implication $\theta(t) : s \Rightarrow \theta(t') : s$. This is an easily checkable condition which is needed to avoid illegal rewrites. For example, equation `[1]` in `HF-SETS` is sort-decreasing, because any instance of `M` must have sort `Set` (and therefore also sort `Magma`); and any instance of `M , M` must have sort `Magma`. Instead, the equation `M = M , M` is *not* sort-decreasing. This can easily lead to illegal rewrite steps. For example, `P(0)` $\to$ `P(0 , 0)` is supposed to be a rewrite step with equation `M = M , M`, but violates Definition 7, and is therefore *illegal*, because `P(0 , 0)` is not a well-formed term.

A third requirement is one of *determinism*: the final result of a reduction with the equations should not depend on the order of evaluation, that is, on the particular order in which the rewrites have been performed in the rewrite sequence. Otherwise, different Maude implementations might produce different results in answer to a `red` command, which would be quite confusing. This is captured by the requirement of *confluence*. Since the relation $u \to_E^* v$ is just the special case of the relation $u \to_{E/B}^* v$ in which $B = \emptyset$, everything can be captured by the following definition:

Definition 8 *Let E and B be sets of Σ-equations. Then the equations E are called* confluent modulo B *(resp.,* ground confluent modulo B*) iff for each $t \in T_{\Sigma(Y)}$ (resp., for each $t \in T_\Sigma$), and each pair of rewrite proofs $t \to_{E/B}^* u$, $t \to_{E/B}^* v$, there is a term $w \in T_{\Sigma(Y)}$ (resp., $w \in T_\Sigma$) such that $u \to_{E/B}^* w$ and $v \to_{E/B}^* w$. This condition can be described diagrammatically as follows (the dashed arrows denote existential quantification):*



Our third requirement of "determinism" is then that: (3) the equations E should be *ground confluent* modulo B. Although ground confluence is a weaker condition than confluence, it is sufficient for declarative programming purposes, since the expressions that we normally submit for reduction are *ground terms* with concrete data, not terms with variables. Maude, however, can also reduce terms with variables.

Exercise 21 *Call two terms $t, t' \in T_{\Sigma(Y)}$* joinable *with E modulo B, denoted $t \downarrow_{E/B} t'$, iff $(\exists w \in T_{\Sigma(Y)})\ t \to_{E/B}^* w \wedge t' \to_{E/B}^* w$. Prove that if the equations E are confluent modulo axioms B, the following equivalence holds for any two terms $t, t' \in T_{\Sigma(Y)}$:*

$$t =_{E \cup B} t' \ \Leftrightarrow\ t \downarrow_{E/B} t'.$$

(Hint: You may want to first prove that $t =_{E \cup B} t' \ \Leftrightarrow\ t \to_{E \cup E^{-1}/B}^* t'$).

It is of course highly desirable that a declarative program terminates, so this should be our next requirement.

Definition 9 *Let E and B be sets of Σ-equations. E is called* terminating *or* strongly normalizing *modulo B (resp.,* ground terminating *or* strongly ground normalizing *modulo B), iff there is no term $t \in T_{\Sigma(Y)}$ (resp., no term $t \in T_\Sigma$), having an infinite sequence of rewrite steps with E modulo B starting from t:*

$$t \to_{E/B} t_1 \to_{E/B} t_2 \ldots t_n \to_{E/B} t_{n+1} \to \ldots$$

We call E weakly terminating *or* normalizing *modulo B (resp.,* ground weakly terminating *or* ground normalizing *modulo B), iff for any $t \in T_{\Sigma(Y)}$ (resp., for any $t \in T_\Sigma$), there is a $t' \in T_{\Sigma(Y)}$ (resp., $t' \in T_\Sigma$) such that $t \to_{E/B}^! t'$.*

Note that if E satisfies condition (1) and Σ has nonempty sorts, it is easy to check that termination is equivalent to ground termination, because if we have a sequence $t \rightarrow_{E/B} t_1 \rightarrow_{E/B} t_2 \ldots t_n \rightarrow_{E/B} t_{n+1} \rightarrow \ldots$ and X is the set of variables appearing in t, we can always find a *ground* substitution $\theta : X \longrightarrow T_\Sigma$ which then yields a nonterminating sequence of ground terms $\theta(t) \rightarrow_{E/B} \theta(t_1) \rightarrow_{E/B} \theta(t_2) \ldots \theta(t_n) \rightarrow_{E/B} \theta(t_{n+1}) \rightarrow \ldots$. Therefore, a highly desirable fourth requirement is: (4) the equations E are terminating modulo B, or at least the weaker requirement (4′) that the equations E are weakly terminating modulo B.

Essentially, conditions (1)–(4) are all we need to require of a functional module fmod $(\Sigma, E \cup B)$ endfm for executability purposes, even though termination, while highly desirable, should not be made into an absolute requirement. It is however very useful methodologically to identify a subsignature $\Omega \subseteq \Sigma$ of *constructor symbols* (with same poset of sorts as Σ). Such a distinction between defined and constructor symbols is precisely what Maude's `ctor` attribute allows. This is useful because we can then ask the question: have we given *enough equations* to *fully define* all functions? This exactly means that the defined function symbols should *disapper* from any ground term after such a term has been fully reduced, so that its reduced form has only constructor symbols.

Definition 10 *Let E and B be sets of Σ-equations, and let $\Omega \subseteq \Sigma$ be a subsignature with same poset of sorts as Σ, that is, $\Sigma = ((S, <), F)$, $\Omega = ((S, <), G)$, and $G \subseteq F$. We say that the equations E are* sufficiently complete modulo *B with respect to the* constructor subsignature *Ω iff for each $s \in S$ and each $t \in T_{\Sigma,s}$ there is a $t' \in T_{\Omega,s}$ such that $t \rightarrow^{!}_{E/B} t'$.*

Therefore, if we have identified for our functional module fmod $(\Sigma, E \cup B)$ endfm a subsignature of Ω of constructors, a fifth and last requirement should be: (5) the equations E are sufficiently complete modulo B.

Condition (1) is syntactically checkable, and so is condition (2). Confluence is a decidable property for $T = (\Sigma, E \cup B)$ when B consists of associativity, and/or commutativity, and/or identity axioms and any associative operator is also commutative. Since term rewriting can simulate Turing machines (see, e.g., [3], 5.1), because of the halting problem termination is undecidable. Sufficient completeness is also decidable for $T = (\Sigma, E \cup B)$ when E is weakly terminating modulo B, and B consists of associativity, and/or commutativity, and/or identity axioms and any associative operator is also commutative, if for any $t = t' \in E$, any variable x of t occurs only once in t (the so-called *left-linearity* of $t = t'$). Maude automatically checks condition (1), can automatically check confluence and condition (2) with its Church-Rosser Checker (CRC) tool [18], and also condition (5) under the above assumptions with its Sufficient Completeness Checker (SCC) tool [25]. Although termination is undecidable, it can be proved in practice for many equational theories using Maude's Termination Tool (MTT) [17]. When the equations E are conditional (see §5.5), both confluence and sufficient completeness become undecidable, but it is still possible to use the above tools and Maude's Inductive Theorem Prover (ITP) to establish such properties for many specifications.

Note that if a theory $T = (\Sigma, E \cup B)$, with $\Omega \subseteq \Sigma$, satisfies conditions (1)–(5), then, by conditions (5), for any $t \in T_{\Sigma,s}$, there is a $t' \in T_{\Omega,s}$ with $t \rightarrow^{!}_{E/B} t'$, and by condition (3), for any other $t'' \in T_\Omega$ with $t \rightarrow^{!}_{E/B} t''$ we *must* have $t' =_B t''$, that is, the results of reduction are *unique* modulo the axioms B. We can make this uniqueness more explicit by defining the *canonical form* of a ground term $t \in T_\Sigma$ by the equations E modulo B, denoted $can(t)$, to be the set $can(t) = \{t' \in T_\Omega \mid t \rightarrow^{!}_{E/B} t'\}$. Note that, by conditions (3) and (5), if $t' \in can(t)$, then $can(t) = \{t'' \in T_\Omega \mid t'' =_B t'\}$, that is, $can(t)$ is what is called a *B-equivalence class* of constructor terms modulo B, where given $t' \in T_\Omega$ we define its *B-equivalence class* as the set of terms $[t']_B = \{t'' \in T_\Omega \mid t'' =_B t'\}$. Note that if the axioms B are a set of associativity and/or commutativity equations, then B-equivalence classes are always *finite*. For example, in the module `HF-SETS`, the AC-equivalence class of the term `{0,{0}}` is: `[{0,{0}}]`$_{AC}$ = `{{0,{0}},{{0},0}}`. We can then define the set of B-equivalence classes[6] as $T_{\Omega/B} = \{[t']_B \mid t' \in T_\Omega\}$. It is easy to prove that for any two terms $t', t'' \in T_{\Omega,s}$ we have $[t']_B \neq [t'']_B$ iff $\neg(t' =_B t'')$ iff $[t']_B \cap [t'']_B = \emptyset$. That is, the set $T_{\Omega/B}$ is a *partition* of the set T_Ω. Assuming that the theory $T = (\Sigma, E \cup B)$ satisfies at least conditions (3) and (5), we can now define the set of *canonical forms* of the equations E modulo B as the set $Can_T = \{can(t) \in T_{\Omega/B} \mid t \in T_\Sigma\}$; and for each sort $s \in S$ we can define the set of canonical forms of sort s as the set $Can_{T,s} = \{can(t) \in T_{\Omega/B} \mid t \in T_{\Sigma,s}\}$.

What is Can_T? it is the set of *values* (up to B-equivalence) computed by reduction using the theory $T = (\Sigma, E \cup B)$. That is, if T is a Maude module satisfying conditions (1)–(5), then Can_T is the set of values

[6] Equivalence relations and equivalence classes with be studied in detail in 9.6. The essentail point is that for any set of equations E, the relation $=_E$ is always an equivalence relation on terms.

that, up to B-equivalence, Maude will return as results of `red` commands. In particular, for any defined function symbol $f : s_1 \ldots s_n \longrightarrow s$ and for any values $([t_1]_B, \ldots, [t_n]_B) \in Can_{T,s_1} \times \ldots \times Can_{T,s_n}$, we can then ask, what is the *result* of evaluating $f([t_1]_B, \ldots, [t_n]_B)$ in T? Let us call such a result $f_T([t_1]_B, \ldots, [t_n]_B)$. It should be, up to B-equivalence, what Maude returns, and therefore should be a value in $Can_{T,s}$. Which value? Precisely the following:

$$f_T([t_1]_B, \ldots, [t_n]_B) = can(f(t'_1, \ldots, t'_n))$$

for $t'_1 \in [t_1]_B, \ldots, t'_n \in [t_1]_B$, such that $t'_1 \in T_{\Omega,s_1}, \ldots, t'_n \in T_{\Omega,s_n}$. That is, we just choose representatives of the appropriate sort for each B-equivalence class and compute the canonical form of the term $f(t'_1, \ldots, t'_n)$. Note that (3), (5), and the definition of $\rightarrow_{E/B}$ ensure that this result *does not depend on the representatives chosen*, so that f_T is well-defined. That is, if we were to choose $t''_1 \in [t_1]_B, \ldots, t''_n \in [t_1]_B$, such that $t''_1 \in T_{\Omega,s_1}, \ldots, t''_n \in T_{\Omega,s_n}$, then if $f(t'_1, \ldots, t'_n) \rightarrow^!_{E/B} u$ and $f(t''_1, \ldots, t''_n) \rightarrow^!_{E/B} u'$, we must have $u =_B u'$, and therefore $can(f(t'_1, \ldots, t'_n)) = can(f(t''_1, \ldots, t''_n))$.

All this is just what we would expect. It gives us a more abstract way of reinterpreting Maude's `red` command. For example, consider the command:

```
Maude> red {{0,{0}},{0}} U  {0,{0},{{0}}} .
result Set: {0,{0},{{0}},{0,{0}}}
```

In our new notation we view this as the performing of the operation $\mathtt{U}_{\mathtt{HF-SETS}}$ on constructor values, so that
$[\mathtt{\{\{0,\{0\}\},\{0\}\}}]_{AC}\ \mathtt{U}_{\mathtt{HF-SETS}}\ [\mathtt{\{0,\{0\},\{\{0\}\}\}}]_{AC} = can(\mathtt{\{\{0,\{0\}\},\{0\}\}\ U\ \{0,\{0\},\{\{0\}\}\}}) = [\mathtt{\{0,\{0\},\{\{0\}\},\{0,\{0\}\}\}}]_{AC}$.

But does `HF-SETS` satisfy conditions (1)–(5)?

Theorem 3 *The theory specified be the functional module* `HF-SETS` *satisfies conditions (1)–(5).*

Proof. Details to be provided.

Exercise 22 *Prove in detail that if $T = (\Sigma, E \cup B)$ satisfies conditions (1)–(5), then if $f : s_1 \ldots s_n \longrightarrow s$ is a defined function symbol and $t'_1, t''_1 \in [t_1]_B, \ldots, t'_n, t''_n \in [t_1]_B$ are such that $t'_1, t''_1 \in T_{\Omega,s_1}, \ldots, t'_n, t''_{1n} \in T_{\Omega,s_n}$, then $can(f(t'_1, \ldots, t'_n)) = can(f(t''_1, \ldots, t''_n))$.*

5.4 A Computable Model of HF-Sets

We are now ready to prove the claim that the Maude module `HF-SETS` provides a computable model of hereditarily finite sets. But what do we mean by a "model"? And what doest it mean for a model to be a model of something? Unless this is clarified, we may be talking nonsense, or using some poetic imagery, or we might perhaps be engaging in software engineering tool salesmanship. The answer to the second question is that a model is a model of a *theory*, in our case, a first-order theory. Which one? Of course, a theory that gives axioms for HF-sets. Which axioms? Since HF-sets are *sets*, the most natural candidate is some fragment of the *ZFC* axioms. But since HF-sets are finite, the axiom of infinity obviously cannot hold. It turns out, however, that HF-sets satisfy *all the axioms of set theory, except the axiom of infinity*. I shall prove this in detail for the other set theory axioms we have encountered so far. When the remaining set theory axioms, namely, (*AC*), (*Rep*), and (*Found*), are explained in in later chapters, their satisfaction in the model of HF-sets will be the subject of several exercises.

But what is a model? What does it mean for a first-order theory T to *have* a model? I answer this question for first-order theories whose languages use only predicate symbols. It turns out that all other first-order theories can be translated into equivalent theories of this kind. Furthermore, set theory is a special case, since its language uses only the binary predicate symbol $\in$. That is, I am assuming a first-order theory $T = (\Pi, \Gamma)$ where Π is a collection of pairs (P, n), with P a predicate symbols and $n \in \mathbb{N}$ its number of arguments, and where Γ is a collection of (fully quantified) formulas in the language of Π. That is, atomic formulas are either $P(x_1, \ldots, x_n)$ for some $(P, n) \in \Pi$, with the $x_1, \ldots, x_n$ variables, or equalities between variables $x = y$, and the rest of the formulas are built out of these atomic formulas in the usual way, by repeated application of Boolean connectives and of universal and existential quantification, as illustrated for the language of set theory in Chaper 2. The axioms Γ are then a collection of such formulas that we assume fully quantified, that is, they are formulas with no free variables.

Suppose a theory $T = (\Pi, \Gamma)$ like this. What is a model of T? A *model of the language of* Π is a pair $(M, \{P_M\}_{(P,n)\in\Pi})$ where:

- M is a nonempty set, and
- for each $(P, n) \in \Pi$, P_M is a subset $P_M \subseteq M^n$.

This provides an *interpretation* for the *symbols* of Π. For example, if Π has just one binary predicate symbol $<$, we may take $M = \mathbb{N}$, the von Neumann natural numbers, and $<_{\mathbb{N}} = \{(n, m) \in \mathbb{N}^2 \mid n \in m\}$.

Such a model $(M, \{P_M\}_{(P,n)\in\Pi})$ is then called a *model of the theory* $T = (\Pi, \Gamma)$ iff, in addition, it *satisfies* the axioms Γ, that is, the axioms Γ should be *theorems* that we can prove hold true for this interpretation of our language. For example, for the language of $\Pi = \{(<, 2)\}$, the axioms Γ could consist of the single formula $(\forall n, m)\ n < m \vee m < n \vee n = m$. We could then rephrase Exercise 19 as asking us to prove that $(\mathbb{N}, \{<_{\mathbb{N}}\})$, is a model of the theory $T = (\{(<, 2)\}, \{(\forall n, m)\ n < m \vee m < n \vee n = m\})$.

Implicit in all this is the fact (see, e.g., [4]) that, once we fix a model $(M, \{P_M\}_{(P,n)\in\Pi})$ of the language of Π, for any fully quantified fomula φ in the language of Π, *either* φ holds in $(M, \{P_M\}_{(P,n)\in\Pi})$ (sometimes expressed by saying that $(M, \{P_M\}_{(P,n)\in\Pi})$ *satisfies* φ, or that φ is a *theorem* of $(M, \{P_M\}_{(P,n)\in\Pi})$), which we denote $(M, \{P_M\}_{(P,n)\in\Pi}) \models \varphi$, *or* φ does not hold in $(M, \{P_M\}_{(P,n)\in\Pi})$, denoted $(M, \{P_M\}_{(P,n)\in\Pi}) \not\models \varphi$. In this notation, we can rephrase the statement that $(M, \{P_M\}_{(P,n)\in\Pi})$ is a model of $T = (\Pi, \Gamma)$ as the statement that $(M, \{P_M\}_{(P,n)\in\Pi}) \models \varphi$ for each $\varphi \in \Gamma$. Instead, if φ is a formula with a single free variable x, what φ determines in a model $(M, \{P_M\}_{(P,n)\in\Pi})$ is not a truth value but a *subset* of M, which we may denote φ_M. For example, for $(\mathbb{N}, \{<_{\mathbb{N}}\})$, the formula $\varphi = x > x$ determines the subset $\varphi_{\mathbb{N}} = \emptyset$, and the formula $\varphi = (\exists y)\ y < x$ determines the subset $\varphi_{\mathbb{N}} = \{n \in \mathbb{N} \mid 0 \in n\}$. More generally, a formula φ with free variables $x_1, \ldots, x_n$ determines in a model $(M, \{P_M\}_{(P,n)\in\Pi})$ a *subset* $\varphi_M \subseteq M^n$. For example, the formula $\varphi = \neg(x_1 < x_2)$ determines in $(\mathbb{N}, \{<_{\mathbb{N}}\})$, the subset $\varphi_{\mathbb{N}} = \{(n, m) \in \mathbb{N}^2 \mid n \notin m\}$.

Therefore, to show that the Maude module `HF-SETS` provides a model of all the set theory axioms we have seen except (*Inf*), that is, of the theory $(\{(\in, 2)\}, \{(\emptyset), (Ext), (Sep), (Pair), (Union), (Pow)\})$, what we need to do is to associate to `HF-SETS` a set, say $\mathscr{V}_\omega$, and a subset $\in_{\mathscr{V}_\omega} \subseteq \mathscr{V}_\omega^2$ interpreting the language of $\{(\in, 2)\}$, and then show that $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models \varphi$ for each $\varphi \in \{(\emptyset), (Ext), (Sep), (Pair), (Union), (Pow)\}$. But remember the claim that `HF-SETS` provides a *computable* model of HF-sets. What does this mean? We need to make explicit the notion of *computable set*. This will serve two puposes. First, it will back the intuition that specifying HF-sets by means of a *program* like `HF-SETS` must of course give us *computable* model of HF-sets. Second, it will also make clear that the notion of computable set we are using is *constructive*, whereas the general notion of set in *ZFC* is not constructive. That is, the notion of model of a first-order theory is *relative* to the notion of set we happen to adopt, and in this case we are adopting a constructive notion.

What is this notion? The notion of computable set I will use, called a *space*, is based on the the axiomatic approach to computability outlined by Shoenfield in [49]. The basic notions are that of a *finite object*, a *space* of finite objects, and a *recursive function*. In Shoenfield's own words, a *finite object* is an "object which can be specified by a finite amount of information;" computer scientists call this a finite *data structure*. For example: (i) a finite order-sorted signature Σ with explicitly-given syntax, as all signatures of Maude modules are, is obviously a finite object; (ii) a Σ-term in such a syntax is also a finite object; and (iii) any canonical form $[t]_{AC} \in Can_{\texttt{HF-SETS}}$ is also a finite object, namely, a finite set of terms that we can effectively generate as soon as we are given one of its representatives. A *space* is "an infinite class X of finite objects such that, given a finite object x, we can decide whether or not x belongs to X." I will ammend Shoenfield a little by saying instead that a *space* is *a countable (finite of infinite) class X of finite objects such that, given a finite object x, we have an algorithm to decide whether or not x belongs to X*. Computer scientists call this a *data type*. Both the sets T_Σ and $T_{\Sigma,s}$ are not just sets but spaces in this sense: we can use the parsing rules in Definition 3 to decide whether $t \in T_\Sigma$, or $t \in T_{\Sigma,s}$. Similarly, $Can_{\texttt{HF-SETS}}$, and also $Can_{\texttt{HF-SETS,Set}}$, $Can_{\texttt{HF-SETS,Magma}}$, and $Can_{\texttt{HF-SETS,Bool}}$ are spaces, since we can decide when a term t is in canonical form modulo AC, and therefore when $[t]_{AC} \in Can_{\texttt{HF-SETS}}$, or it belongs to any of the subsets mentioned. Note that if $A_1, \ldots, A_n$ are spaces, $A_1 \times \ldots \times A_n$ is also space, since n-tuples of finite objects are themselves finite objecs, and we can decide membership in $A_1 \times \ldots \times A_n$ by deciding membership in each A_i for each component of a tuple. Given spaces X and Y, a *recursive function*[7] $f : X \longrightarrow Y$ is then a (total!) function that can be computed by an *algorithm*, i.e., by a computer

[7] Functions will be studied in detail in §6; and we will discuss various kinds of recursive functions in later chapters. For the moment,

program, when we disregard space and time limitations. For example, the process of computing canonical forms by rewriting supported in Maude by the `red` command gives us an algorithm to compute the recursive function $can : T_{\Sigma_{\texttt{HF-SETS}}} \longrightarrow Can_{\texttt{HF-SETS,Set}}$, where $\Sigma_{\texttt{HF-SETS}}$ denotes the signature of the `HF-SETS` module. Similarly, for each defined function symbol f in `HF-SETS`, the functions $f_{\texttt{HF-SETS}}$ are recursive functions. For example, $\texttt{U}_{\texttt{HF-SETS}}$ is a recursive function $\texttt{U}_{\texttt{HF-SETS}}$: $Can^2_{\texttt{HF-SETS,Set}} \longrightarrow Can_{\texttt{HF-SETS,Set}}$, and $\texttt{in}_{\texttt{HF-SETS}}$ is a recursive function $\texttt{in}_{\texttt{HF-SETS}}$: $Can_{\texttt{HF-SETS,Magma}} \times Can_{\texttt{HF-SETS,Set}} \longrightarrow Can_{\texttt{HF-SETS,Bool}}$. Note that if $f : X \longrightarrow Y$ is a recursive function, the choice of a finite object $y \in Y$ defines what I will call a *subspace* of X, namely, the space, denoted $f^{-1}[\{y\}]$, which is defined by the equality $f^{-1}[\{y\}] = \{x \in X \mid f(x) = y\}$.

At last, we are now ready to say in which sense `HF-SETS` provides for us a *computable* model of the language of $\{(\in, 2)\}$. First of all, we choose as our set $\mathscr{V}_\omega$ the *space* $Can_{\texttt{HF-SETS,Set}}$, second, we interpret the predicate $\in$ as the *subspace* $\in_{\mathscr{V}_\omega} \subseteq \mathscr{V}_\omega^2$ defined by the equality $\in_{\mathscr{V}_\omega} = \texttt{in}_{\texttt{HF-SETS}}{}^{-1}[\{[\texttt{true}]\}]$. That is, simplifying $[t]_{AC}$ to just $[t]$, considering $[s], [s'] \in Can_{\texttt{HF-SETS,Set}}$, and adopting an infix notation for $\in_{\mathscr{V}_\omega}$ by means of the defining equivalence $[s] \in_{\mathscr{V}_\omega} [s'] \;\Leftrightarrow\; ([s], [s']) \in (\in_{\mathscr{V}_\omega})$, we have:

$$[s] \in_{\mathscr{V}_\omega} [s'] \;\Leftrightarrow\; \texttt{in}_{\texttt{HF-SETS}}([s], [s']) = [\texttt{true}] \;\Leftrightarrow\; can(s\,\texttt{in}\,s') = [\texttt{true}].$$

The culmination of this entire chapter is the following theorem:

Theorem 4 *$(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\})$ is a model of the theory $(\{(\in, 2)\}, \{(\varnothing), (Ext), (Sep), (Pair), (Union), (Pow)\})$.*

Proof. I will use, e.g., $[s], [s'], [\{s, m\}]$, and so on, to denote elements $[s], [s'], [\{s, m\}] \in \mathscr{V}_\omega$. Likewise, defining $\mathscr{M}_\omega = Can_{\texttt{HF-SETS,Magma}}$, I will use, e.g., $[m], [m'], [m, m'], [s, m]$, and so on, to denote elements $[m], [m'], [m, m'], [s, m] \in \mathscr{M}_\omega$. Let us first prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\varnothing)$, that is, $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\exists x)(\forall y)\; y \notin x$. Of course, we choose $[\texttt{0}]$ for the existentially quantified x, and we have to show that for each $[s] \in \mathscr{V}_\omega$ $([s], [\texttt{0}]) \notin (\in_{\mathscr{V}_\omega})$, that is, that $can(s\,\texttt{in}\,[\texttt{0}]) = [\texttt{false}]$, which follows trivially from equation `[2]`.

Before proving that the other axioms are also satisfied we need a lemma.

Lemma 2 *For each $[s], [s'] \in \mathscr{V}_\omega$, the following two properties hold:*

1. $[s] \in_{\mathscr{V}_\omega} [s'] \;\Leftrightarrow\; [s'] = [\{s\}] \vee (\exists [m] \in \mathscr{M}_\omega)\; [s'] = [\{s, m\}]$
2. $[s] = [s'] \;\Leftrightarrow\; can(s \sim s') = [\texttt{true}]$.

Proof. I will prove the lemma using a function $size : \mathscr{V}_\omega \longrightarrow \mathbb{N}$, defined by the equalities: $size([\texttt{0}]) = 1$, $size([\{s\}]) = 1 + size([s])$, and $size([\{s, m\}]) = size([s]) + size([\{m\}])$. We can then define the measure function $\mu : \mathscr{V}_\omega^2 \longrightarrow \mathbb{N}$ by means of the equality $\mu([s], [s']) = size([s]) + size([s'])$. I reason by contradiction. Suppose that the lemma fails, and let $[s], [s'] \in \mathscr{V}_\omega$ be a pair of elements for which the lemma fails with $\mu([s], [s'])$ smallest possible. Note that we cannot have $\mu([s], [s']) = 0$, since then $[s] = [s'] = [\texttt{0}]$, and it is easy to check, using the definition of $[s] \in_{\mathscr{V}_\omega} [s']$ and the equations in `HF-SETS`, that both (1) and (2) hold. Furthermore, we cannot have $[s'] = [\texttt{0}]$, since then either $[s]$ is of the form $[\{s''\}]$ or of the form $[\{s'', m\}]$, and then (1) trivially holds, because of equation `[2]` and the fact that the disjunction $[s'] = [\{s\}] \vee (\exists [m] \in \mathscr{M}_\omega)\; [s'] = [\{s, m\}]$ cannot hold; and (2) trivially holds because $[s] \neq [\texttt{0}]$, and for both cases, $[s] = [\{s''\}]$ and $[s] = [\{s'', m\}]$, the equations in `HF-SETS` force $can(s \sim \texttt{0}) = [\texttt{false}]$. Therefore, we must have $[s]$ of the form $[\{s_1, \ldots, s_n\}]$ and $[s']$ of the form $[\{s'_1, \ldots, s'_m\}]$, with $n, m > 1$ and either (1) or (2) failing. I will prove that (2) failing leads to a contradiction, and leave the proof that (1) failing leads to a similar contradiction as an exercise. If (2) fails, then either (a) $[\{s_1, \ldots, s_n\}] = [\{s'_1, \ldots, s'_n\}]$, and $can(\{s_1, \ldots, s_n\} \sim \{s'_1, \ldots, s'_n\} = [\texttt{false}]$, or (b) $[\{s_1, \ldots, s_n\}] \neq [\{s'_1, \ldots, s'_n\}]$, and $can(\{s_1, \ldots, s_n\} \sim \{s'_1, \ldots, s'_n\} = [\texttt{true}]$. But in case (a), the fact that $[\{s_1, \ldots, s_n\}]$ and $[\{s'_1, \ldots, s'_n\}]$ are in canonical form and are equal modulo AC means that there are no repeated elements, $n = m$, and there is a permutation mapping i, $1 \leq i \leq n$, to j_i such that $[s_i] = [s'_{j_i}]$. But this leads to a contradiction, because the assumption that $\mu([s], [s'])$ is smallest possible such that the lemma fails means that for $1 \leq i \leq n$ we must have $can(s_i \sim s'_{j_i}) = [\texttt{true}]$, and the equations in `HF-SETS` then force $can(\{s_1, \ldots, s_n\} \sim \{s'_1, \ldots, s'_n\}) = [\texttt{true}]$. Similarly, case (b) leads to a contradiction since, again, the assumption that $\mu([s], [s'])$ is smallest possible such that the lemma fails means that for each $1 \leq i \leq n$, $1 \leq j \leq m$, we have $[s_i] = [s'_j] \;\Leftrightarrow\; can(s_i \sim s'_j) = [\texttt{true}]$. But then the lack of repeated elements in

the notion that a recursive function $f : X \longrightarrow Y$ is a rule of computation such for each input $x \in X$ its result $f(x) \in Y$ is computed by a terminating computer program is perfectly adequate.

$[\{s_1, \ldots, s_n\}]$ and $[\{s'_1, \ldots, s'_n\}]$, and the equations in HF-SETS force that for each i, $1 \leq i \leq n$, there is a unique j_i, $1 \leq j_i \leq m$, with $[s_i] = [s'_{j_i}]$, and for each j, $1 \leq j \leq m$ there is a unique i_j, $1 \leq i_j \leq n$ with $[s'_j] = [s_{i_j}]$. And this forces the contradiction $n = m$ and $[\{s_1, \ldots, s_n\}] = [\{s'_1, \ldots, s'_n\}]$, as desired. □

Let us now prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (Ext)$, that is, $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x, y)((\forall z)(z \in x \Leftrightarrow z \in y) \Rightarrow x = y)$. Suppose that $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \not\models (Ext)$. This means that we have $[s], [s'] \in \mathscr{V}_\omega$ such that for each $[s''] \in \mathscr{V}_\omega$ we have $[s''] \in_{\mathscr{V}_\omega} [s] \Leftrightarrow, [s''] \in_{\mathscr{V}_\omega} [s']$, but $[s] \neq [s']$. Therefore, if $[s] = [\mathtt{0}]$, then $[s']$ must be of the form $[\{s_1\}]$ or of the form $[\{s_1, m\}]$. Either way, by choosing $[s''] = [s_1]$ and using Lemma 2 and equation [2] we get that $[s_1] \in_{\mathscr{V}_\omega} [\mathtt{0}]$ does not hold, but $[s_1] \in_{\mathscr{V}_\omega} [s']$ does hold, contradicting the equivalence $[s''] \in_{\mathscr{V}_\omega} [s] \Leftrightarrow, [s''] \in_{\mathscr{V}_\omega} [s']$. Therefore, we must have $[s]$ of the form $[\{s_1, \ldots, s_n\}]$ and $[s']$ of the form $[\{s'_1, \ldots, s'_m\}]$, with $n, m > 1$. But then then choices $[s''] = [s_i]$, $1 \leq i \leq n$ and the equivalence $[s_i] \in_{\mathscr{V}_\omega} [s] \Leftrightarrow, [s_i] \in_{\mathscr{V}_\omega} [s']$ force $[s_i] \in_{\mathscr{V}_\omega} [s']$, and therefore by Lemma 2 there is a unique j_i, $1 \leq j_i \leq m$ such that $[s_i] = [s'_{j_i}]$. The same reasoning with choices $[s''] = [s'_j]$, $1 \leq j \leq m$ force that for each j there is a unique i_j, $1 \leq i_j \leq n$ such that $[s'_j] = [s_{i_j}]$; and this forces $n = m$ and $[\{s_1, \ldots, s_n\}] = [\{s'_1, \ldots, s'_m\}]$, contradicting the assumption $[s] \neq [s']$.

Let us now prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (Sep)$, that is, for each φ whose only free variable is x we have to prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall y)(\exists! z)(\forall x)(x \in z \Leftrightarrow (x \in y \wedge \varphi))$. Since we have already proved $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (Ext)$, it is enough to prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall y)(\exists z)(\forall x)(x \in z \Leftrightarrow (x \in y \wedge \varphi))$. We have to consider two cases. If $y = [\mathtt{0}]$, then, by equation [2], the choice $z = [\mathtt{0}]$ makes the implication true for all x. Otherwise, y must be of the form $[\{s_1, \ldots, s_n\}]$, $n \geq 1$. Now recall that any formula φ whose only free variable is x defines a subset $\varphi_{\mathscr{V}_\omega} \subset \mathscr{V}_\omega$. Let $[s_{i_1}], \ldots, [s_{i_k}]$, $1 \leq i_1 < \ldots < i_k$, be the elements among the $[s_1], \ldots, [s_n]$ such that $[s_{i_j}] \in \varphi_{\mathscr{V}_\omega}$. Then, by Lemma 2, the choice $z = [\{s_{i_1}, \ldots, s_{i_k}\}]$ makes the formula true, since for any x, $x \in z$ iff $x = [s_{i_j}]$ for some $1 \leq j \leq k$, and then clearly $(x \in y \wedge \varphi)$ must hold, since $[s_{i_j}] \in \varphi_{\mathscr{V}_\omega}$.

Let us prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (Pair)$, that is, $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x, y)(\exists! z)(\forall u)(u \in z \Leftrightarrow (u = x \vee u = y))$. Again, it is enough to prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x, y)(\exists z)(\forall u)(u \in z \Leftrightarrow (u = x \vee u = y))$. We again reason by cases. If $x = y = [s]$, we choose $z = [\{s\}]$. Therefore, by Lemma 2, $u \in z$ iff $u = [s]$ iff $(u = x \vee u = y)$, as desired. Otherwise, $x = [s]$, $y = [s']$, and $[s] \neq [s']$. We then choose $z = [\{s, s'\}]$, which is a term in canonical form. And, again, Lemma 2 forces the equivalence $(\forall u)(u \in z \Leftrightarrow (u = x \vee u = y))$.

Let us prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (Union)$, that is, $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x)(\exists! y)(\forall z)(z \in y \Leftrightarrow (\exists u)(u \in x \wedge z \in u))$. Again, it is enough to prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x)(\exists y)(\forall z)(z \in y \Leftrightarrow (\exists u)(u \in x \wedge z \in u))$. For the case $x = [\mathtt{0}]$, it is easy to check using Lemma 2 that the choice $y = [\mathtt{0}]$ makes the formula true. Otherwise, x is either (i) $x = [\{\mathtt{0}\}]$, or (ii) of the form $x = [\{\mathtt{0}, \{m_1\}, \ldots, \{m_n\}\}]$, $n \geq 1$, or (iii) of the form $x = [\{\{m_1\}, \ldots, \{m_n\}\}]$, $n \geq 1$. In case (i) the choice $y = [\mathtt{0}]$ makes the formula true. In cases (ii) and (iii), one easily proves using Lemma 2 that the choice $y = can(\{m_1, \ldots, m_n\})$ makes the formula true.

Let us finally prove that $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (Pow)$, that is, $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x)(\exists! y)(\forall z)(z \in y \Leftrightarrow z \subseteq x)$. Again, it is enough to prove $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\}) \models (\forall x)(\exists y)(\forall z)(z \in y \Leftrightarrow z \subseteq x)$. For $x = [s]$, the choice of y is of course $y = can(\mathtt{P}(s))$. To prove that this choice makes the formula true, we need some auxiliary notation and facts. Recall that the formula $x_1 \subseteq x_2$ is auxiliary notation for the formula $(\forall x)\ x \in x_1 \Rightarrow x \in x_2$. Recall also that the formula $x_1 \subseteq x_2$ defines a subset of pairs in our model $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\})$, namely, the subset $(x_1 \subseteq x_2)_{\mathscr{V}_\omega} \subseteq \mathscr{V}_\omega^2$, abbreviated $\subseteq_{\mathscr{V}_\omega}$, defined by the equality, $\subseteq_{\mathscr{V}_\omega} = \{([s], [s']) \in \mathscr{V}_\omega^2 \mid (\forall x \in \mathscr{V}_\omega)\, x \in_{\mathscr{V}_\omega} [s] \Rightarrow x \in_{\mathscr{V}_\omega} [s']\}$. Below, the infix notation $[s] \subseteq_{\mathscr{V}_\omega} [s']$ will abbreviate $([s], [s']) \in \subseteq_{\mathscr{V}_\omega}$. Since the equations for P use the operations _U_ and augment as auxiliary functions, we need some basic facts about such functions, which are gathered in the following lemma, whose proof uses Lemma 2, follows its same pattern of reasoning, and is left as an exercise.

Lemma 3 *The following results hold for $\mathscr{V}_\omega$:*

1. *For each $[s], [s'] \in \mathscr{V}_\omega$, $[s] \subseteq_{\mathscr{V}_\omega} [s']$ iff*
   - *either $[s] = [\mathtt{0}]$, or*
   - *$[s] = [s']$, or*
   - *$(\exists [m], [m'] \in \mathscr{M}_\omega)\ [s] = [\{m\}] \wedge [s'] = [\{m, m'\}]$.*
2. *For each $[s], [s'] \in \mathscr{V}_\omega$, $[s] \subseteq_{\mathscr{V}_\omega} [s'] \Leftrightarrow can(s <= s') = [\mathtt{true}]$.*
3. *For each $[s], [s'], [s''] \in \mathscr{V}_\omega$, $[s''] \in_{\mathscr{V}_\omega} can(s \cup s') \Leftrightarrow [s''] \in_{\mathscr{V}_\omega} [s] \vee [s''] \in_{\mathscr{V}_\omega} [s']$.*

4. *For each* $[s] \in \mathscr{V}_\omega$*, and each* $[\{\mathbb{0}, \{m_1\}, \ldots, \{m_n\}\}] \in \mathscr{V}_\omega$*, with* $n \geq 1$*, and such that, for* $1 \leq i \leq n$*,* $\neg([s''] \in_{\mathscr{V}_\omega} [\{m_i\}])$*, then* $can(\mathtt{augment}(\{\mathbb{0}, \{m_1\}, \ldots, \{m_n\}\}, s)) = [\{\{s\}, \{s, m_1\}, \ldots, \{s, m_n\}\}]$. □

Using Lemma 3, the proof that for $x = [s]$, the choice $y = can(\mathtt{P}(s))$ shows that $\mathscr{V}_\omega \models (\forall x)(\exists y)(\forall z)(z \in y \Leftrightarrow z \subseteq x)$ holds is now relatively simple. We reason by contradiction. Suppose that for some $x = [s]$ the claimed satisfaction fails, and let $size([s])$ be smallest possible among such failing choices of x. Since for $x = [\mathbb{0}]$ and $x = [\{s\}]$ it is easy to prove using equations `[11]` and `[12]` that the claim holds, x must be of the form $x = [\{s, m\}]$. Therefore, $size([\{m\}]) < size([\{s, m\}])$, so that for any $[s'] \in \mathscr{V}_\omega$ we have, $[s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{m\})) \Leftrightarrow [s'] \subseteq_{\mathscr{V}_\omega} [\{m\}]$. We will be finished if we prove that $x = [\{s, m\}]$ was not, after all, a failing choice, that is, that for any $[s'] \in \mathscr{V}_\omega$ we have, $[s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{s, m\})) \Leftrightarrow [s'] \subseteq_{\mathscr{V}_\omega} [\{s, m\}]$. But by the ground confluence of `HF-SETS`, the definition of *can*, equation `[13]`, and Lemma 3 (3), we have the following chain of equivalences: $[s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{s, m\})) \Leftrightarrow [s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{m\}) \,\mathtt{U}\, \mathtt{augment}(\mathtt{P}(\{m\}), s)) \Leftrightarrow [s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{m\})) \vee [s'] \in_{\mathscr{V}_\omega} can(\mathtt{augment}(\mathtt{P}(\{m\}), s))) \Leftrightarrow [s'] \subseteq_{\mathscr{V}_\omega} [\{m\}] \vee [s'] \in_{\mathscr{V}_\omega} can(\mathtt{augment}(\mathtt{P}(\{m\}), s)))$. But using Lemma 3 (1), the fact that $[s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{m\})) \Leftrightarrow [s'] \subseteq_{\mathscr{V}_\omega} [\{m\}]$ means that $can(\mathtt{P}(\{m\}))$ is of the form $can(\mathtt{P}(\{m\})) = [\{\mathbb{0}, \{m_1\}, \ldots, \{m_n\}\}]$, with $can(\{m_1, \ldots, m_n\}) = [\{m\}]$. Therefore, reasoning by cases on whether or not $[s'] \subseteq_{\mathscr{V}_\omega} [\{m\}]$, and using Lemma 3 (4), we get our desired contradiction $[s'] \subseteq_{\mathscr{V}_\omega} [\{s, m\}] \Leftrightarrow [s'] \in [\{\mathbb{0}, \{m_1\}, \ldots, \{m_n\}\}] \vee [s'] \in [\{\{s\}, \{s, m_1\}, \ldots, \{s, m_n\}\}] \Leftrightarrow [s'] \in_{\mathscr{V}_\omega} can(\mathtt{P}(\{s, m\}))$. □

5.5 HF-Sets as a Universe for Finitary Mathematics

It is nice to know that `HF-SETS` is indeed a computable model of HF-sets. But what can you do with it? Why, of course, use it! Since it is indeed a computable model, and set theory is a universal,language for mathematics, we can represent within it all effective constructions of *finitary* mathematics. For example, we can represent within it all effective arithmetic constructions (see [11],Ch. I). How so? Of course, by *defining* those constructions equationally in Maude functional modules that extend `HF-SETS`. This may not necessarily be the most *user-friendly* representation of a particular concept in finitary mathematics: we may have simpler, more direct representations in equational logic itself and can program them in Maude directly without having to *encode* them in `HF-SETS`. In this sense, equational logic can be viewed as a more flexible alternative to set theory to express finitary mathematics in a fully constructive way. Nevertheless, it is conceptually very pleasing to know that we can encode all of finitary mathematics within HF-sets.

How can all this be done in our `HF-SETS` model? Given any concept in finitary mathematics, for example, that of natural number, or that of finite graph, we should first *carve out* the appropriate data elements exemplifying that concept as a *subset* of $\mathscr{V}_\omega$. This can be done by defining a *subsort* of the sort `Set`. Then the appropriate constructions on that domain of thought, for example, natural number addition, or computing the transitive closure of a finite graph, can all be *equationally-defined* by ground confluent and terminating equations, just as we have done on `HF-SETS` for the basic set-theoretic constructions. But can you always do this? *Yes* if your concepts and constructions are *effective*; and they better be effective if your finitary mathematics is going to be any good. But how can you do it? Here it is good to recall our notion of a computable set as a *space*, and of computable function as a *recursive function*. All being effective means exactly that the data elements exemplifying a given concept, e.g., finite graph, form a space; and that the corresponding effective constructions are then recursive functions on that space. Ok, but how do you *know* that you can do all this in equational logic? There are two answers to this question. A first answer is that term rewriting is Turing-complete, and since all effective constructions can be programmed on a computer, we can of course program them with equations. This answer is correct as far as it goes, but it fails to tell us why we could do it with *ground confluent and terminating* equations. A better answer is that *any* (total) recursive function can be programmed as a function f defined in an equational theory (Σ, E) with Σ and E finite, and with the equations E confluent and terminating [6].

So, let's do it! Of course, all of finitary mathematics would not fit in one page. What I can do is to give some examples of how you do it. Other examples will be the subject of subsequent exercises. But it is worth bearing in mind that there is a tension between HF-sets as a universe were *in principle* we can express all of finitary mathematics, and what we might call the *representational cost* of encoding such notions in set theory. In this sense, a subliminal message conveyed by the `HF-SETS` module, and by the `HF-SETS-EXT` module below, is that the representational cost of expressing concepts in equational logic is minimal. Instead, it is quite cumbersome to encode everything in set theory. So, why not using equational

logic directly (together with first-order and inductive reasoning for equational theories) instead of set theory to effectively represent finitary mathematics? Why not indeed! Thoralf Skolem would have probably agreed (see [52] 302–333).

```
fmod HF-SETS-EXT is protecting HF-SETS .
  sorts NeSet Empty One vNat vNzNat ZNat ZNzNat .
  subsorts Empty < vNat ZNat < Set .
  subsorts  One < vNzNat ZNzNat < NeSet < Set .
  subsort vNzNat < vNat .                    subsort ZNzNat < ZNat .
  op 0 : -> Empty [ctor] .
  op {_} : Magma -> NeSet [ctor] .
  op {_} : ZNat -> ZNzNat [ctor] .     *** Zermelo number successor
  op {_} : Empty -> One [ctor] .       *** defines sort One with {0} : One
  var M : Magma .  vars S S' : Set .  var I : vNat .  vars J K : ZNat .

  op \/_ : Set -> Set .            *** general union
  eq [1]: \/ 0 = 0 .
  eq [2]: \/ {S} = S .
  eq [3]: \/ {S,M} = S U \/ {M} .

  op _&_ : Set Set -> Set .       *** intersection
  eq [4]: 0 & S = 0 .
  ceq [5]: {S} & S' = {S} if S in S' = true .
  ceq [6]: {S} & S' = 0 if S in S' = false .
  ceq [7]: {S,M} & S' = {S} U ({M} & S') if S in S' = true .
  ceq [8]: {S,M} & S' = {M} & S' if S in S' = false .

  op /\_ : NeSet -> Set .          *** general intersection
  eq [9]: /\ {S} = S .
  eq [10]: /\ {S,M} = S & /\ {M} .

  op s : Set -> Set .          *** successor function
  op s : vNat -> vNzNat .      *** successor on von Neumann numbers
  eq [10]: s(0) = {0} .
  eq [11]: s({M}) = {M,{M}} .

  cmb [12]: {M,{M}} : vNzNat if {M} : vNzNat .  *** defining the sort vNzNat

  op vplus : vNat vNat -> vNat .     *** von Neumann number addition
  eq [13]: vplus(I,0) = I .
  eq [14]: vplus(I,{0}) = s(I) .
  ceq [15]: vplus(I,{M,{M}}) = s(vplus(I,{M})) if {M} : vNat .

  op Zplus : ZNat ZNat -> ZNat .      *** Zermelo number addition
  eq [16]: Zplus(J,0) = J .
  eq [17]: Zplus(J,{K}) ={Zplus(J,K)} .
 endfm
```

The above module `HF-SETS-EXT` extends `HF-SETS-EXT` with several set theory constructions: the general union operation $\bigcup$, binary intersection $\cap$, and general intersection $\bigcap$. It also introduces two different set-theoretic representations of the natural numbers we have previously encountered: the von Neumann natural numbers $\mathbb{N}$, and the Zermelo natural numbers; and defines the succesor and addition functions for both.

The first thing we need to do is to carve out our new concepts as subsorts of the sort `Set`. Thus, the von Neumann natural numbers are characterized by a sort `vNat`, and the Zermelo natural numbers by a sort `ZNat`, and nonzero natural numbers in each representation by respective subsorts `vNzNat` and `ZNzNat`, which are themselves subsorts of the sort `NeSet` of nonempty sets. Since zero and one have the same representation for von Neumann and Zermelo natural numbers, namely, `0` and `{0}`, both can share a subsort `Empty`, whose only term is the empty set `0`, and a subsort `One`, whose only term is the singleton set `{0}`. All this typing is achieved not just by the above sort and subsort declarations, but by *overloading constructor declarations*. In an order-sorted signature $\Sigma = (S, <, F)$, the same function symbol f can be *overloaded*, that is, it can have several typings in F, say, $(f, s_1 \ldots s_n, s), (f, s'_1 \ldots s'_n, s) \in F$. This allows us to use the same symbol f for an operation that is just the restriction to subsorts of an operation f. For example, by giving the declaration `op 0 : -> Empty [ctor]`, we specify that `Empty` has `0` as its only term. And the three, increasingly tighter overloadings of the constructor `{_}` declare: (i) that nonempty sets are terms of the form `{M}`, (ii) that `{_}` is

the successor function for Zermelo natural numbers, and (iii) that `{0}` is the only term of sort `One`.

Let us see some reductions for the general union operator $\bigcup$, here denoted `\/_`.

```
Maude> red \/ {0,{0,{0}},{{{0}},{0,{0}}}} .
result NeSet: {0,{0},{{0}},{0,{0}}}

Maude> red \/ ({0,{0,{0}},{{{0}},{0,{0}}}} U {0,{0},{{0}},{0,{0}}}) .
result NeSet: {0,{0},{{0}},{0,{0}}}
```

The definition of binary intersection $\cap$, here denoted `_&_`, is interesting, because it is our first encounter with *conditional equations*. These are formulas of the form $(u_1 = v_1 \wedge \ldots \wedge u_n = v_n) \Rightarrow t = t'$, which are implicitly universally quantified, and where the conjunction $(u_1 = v_1 \wedge \ldots \wedge u_n = v_n)$ is called the *condition* of the conditional equation. In Maude they are declared with the `ceq` keyword. How do we evaluate conditional equations by rewriting? If the $v_1, \ldots, v_n$ are terms in canonical form, as it happens for the conditional equations `[5]`–`[8]` for `_&_`, what we do after we find a subterm decomposition of the form $C[\theta(t)]$ is to compute the canonical forms of the $\theta(u_1), \ldots, \theta(u_n)$ and check if they are respectively equal (modulo the structural axioms B) to the $v_1, \ldots, v_n$. If this is so, we have checked that the condition is satisfied, and we can then perform the rewrite $C[\theta(t)] \longrightarrow C[\theta(t')]$. Of course, computing the canonical forms of the $\theta(u_1), \ldots, \theta(u_n)$ may itself require using other conditional equations, and therefore computing also the canonical forms of the corresponding instances in their conditions. Therefore, performing a single step of conditional rewriting involves a *recursive* process of evaluating other possibly conditional rules and their conditions. Here are some reductions for the `_&_` operator:

```
Maude> red {{0,{0}},{{{0}},{0}},{{0}}} & {0,{{0}}} .
result ZNzNat: {{{0}}}

Maude> red {{0,{0}},{{{0}},{0}},{{0}}} & {{0}} .
result Empty: 0
```

Note that the definition of the general intersection operator $\bigcap$, here denoted `/\_`, in terms of the binary operator `_&_` follows a pattern entirely similar to that of `\/_` in terms of `_U_`. Let us see some reductions:

```
Maude> red /\ {{0,{0}},{{{0}},{0}},{{0}}} .
result ZNzNat: {{0}}

Maude> red /\ ({{0,{0}},{{{0}},{0}},{{0}}} & {{0}}) .
result [Magma]: /\ 0
```

The second reduction is interesting. Since `{{0,{0}},{{{0}},{0}},{{0}}} & {{0}}` evaluates to the empty set `0`, the operation `/\_` is not defined on `0`, so we get the expression `/\ 0` of sort `[Magma]` as an error message. Of course, since the term `{{0,{0}},{{{0}},{0}},{{0}}} & {{0}}` has sort `Set` and `/\_` expects a term of sort `NeSet` for its argument, the term `/\ ({{0,{0}},{{{0}},{0}},{{0}}} & {{0}})` is not even well-formed. However, Maude automatically adds an error supersort (called a "kind") on top of each connected component of the sort poset (in this case the kind `[Magma]` on top of `Magma`), and gives such ill-formed expression the benefit of the doubt at evaluation time. If everything goes well, we get an expression with a proper sort; otherwise we get an informative error expression at the kind level.

We next define the successor operation `s` on sets, exactly as defined in §4.2. Of course, since the von Neumann natural numbers are defined in §4.4 as the smallest successor set, we have an oveloaded version of `s` with typing `op s : vNat -> vNzNat .` But note that in our set-theoretic representation of the von Neumann natural numbers `s` is not a constructor, but a defined symbol that eventually disappears of all ground set-theoretic expressions. This poses a challenge. How can we *type* von Neumann natural numbers? That is, given a set $[u] \in \mathscr{V}_\omega$ how can we *decide* whether or not $[u]$ is a von Neumann natural number? If we do not have such a method, the von Neumann natural numbers would not form a space. Note that this is not a problem for Zermelo natural numbers, since their successor function `{_}` is a constructor. Meeting this challenge brings us to a richer equational logic, namely, *membership equational logic* [41], which is actually the logic of Maude functional modules. The idea is that we have two kinds of predicates: the built-in equality predicate = as usual, and for each sort $s \in S$ a unary predicate to test whether an element has sort s, which we write in postfix notation as $\_ : s$. A *membership equational theory* is then of the form $(\Sigma, E \cup B)$, where B are our chosen structural axioms, and where E consists of (possibly conditional) equations and memberships. An unconditional *membership*, specified in Maude with the `mb` keyword, is an

atomic formula of the form $t : s$ (which we assume universally quantified), where t must have at least the kind of s. A *conditional membership*, specified in Maude with the `cmb` is a formula of the form

$$(w_1 : s_1 \ \wedge \ldots \wedge \ w_m : s_m \ \wedge \ u_1 = v_1 \ \wedge \ldots \wedge \ u_n = v_n) \ \Rightarrow \ t : s,$$

which again we assume universally quantified. Furthermore, conditions of conditional equations may now be a conjuntion of equations and memberships, that is, they can be conjunctions of the general form $(w_1 : s_1 \ \wedge \ldots \wedge \ w_m : s_m \ \wedge \ u_1 = v_1 \ \wedge \ldots \wedge \ u_n = v_n)$. Conditional equation `[15]` illustrate this greater generality. The challenge of deciding when $[u] \in \mathscr{V}_\omega$ is a von Neumann natural number is already taken care of for `0` and `{0}` by the sorts `Empty` and `One`, and is met for all other natural numbers by the conditional membership `[12]`. How do we *compute* with conditional memberships? In a Horn-clause, Prolog-like fashion: we match the head of the clause to a given goal and then try to solve the subgoals in the condition to find a proof. Let us illustrate this with an example trace to type the term `{0,{0}}` corresponding to the number 2.

```
Maude> set trace on .
Maude> red {0,{0}} .
reduce in HF-SETS-EXT : {0,{0}} .
*********** trial #1
cmb {M,{M}} : vNzNat if {M} : vNat [label 12] .
M --> 0
*********** solving condition fragment
{M} : vNat
*********** success for condition fragment
{M} : vNat
M --> 0
*********** success #1
*********** membership axiom
cmb {M,{M}} : vNzNat if {M} : vNat [label 12] .
M --> 0
NeSet: {0,{0}} becomes vNzNat
result vNzNat: {0,{0}}
```

Explaining in full detail how the notions of rewriting, confluence, termination, and sufficient completeness generalize from the unconditional order-sorted setting discussed in §5.2 and §5.3 to the conditional membership equational logic setting is beyond the scope of these notes; detailed explanations can be found in [8, 16, 26].

The `vplus` function performs addition of von Neumann natural numbers with the usual recursive definition. Making equation `[15]` conditional ensures that `vplus` will only be performed on sets that are natural numbers. Let us see some reductions.

```
Maude> red vplus(s(s(0)),s(s(0))) .
result vNzNat: {0,{0},{0,{0}},{0,{0},{0,{0}}}}

Maude> red vplus(s(s(0)),{{{0}}}) .
result [Magma]: vplus({0,{0}}, {{{0}}})
```

Finally, `Zplus` performs addition of Zermelo natural numbers. Since this is a constructor-based definition, it is somewhat simpler than that for `vplus`. Again, `Zplus` will only add sets that are Zermelo naturals. Let us see some reductions.

```
Maude> Maude> red Zplus({{0}},{{0}}) .
result ZNzNat: {{{{0}}}}

Maude> red Zplus({{0}},s(s(s(0)))) .
result [Magma]: Zplus({{0}}, {0,{0},{0,{0}}})
```

Exercise 23 *Prove that* `\/_`, `_&_`, *and* `/\_` *are indeed general union, binary intersection, and general intersection. That is, that:*

1. $(\forall [u], [u'] \in \mathscr{V}_\omega)\ ([u'] \in_{\mathscr{V}_\omega} can(\backslash\!/ u) \ \Leftrightarrow \ (\exists [v] \in_{\mathscr{V}_\omega} [u])\ [u'] \in_{\mathscr{V}_\omega} [v])$.
2. $(\forall [u], [u'][v] \in \mathscr{V}_\omega)\ ([v] \in_{\mathscr{V}_\omega} can(u \,\&\, u') \ \Leftrightarrow \ ([v] \in_{\mathscr{V}_\omega} [u] \ \wedge \ [v] \in_{\mathscr{V}_\omega} [u']))$.
3. $(\forall [u] \in \mathscr{V}_\omega - \{[\mathbf{0}]\})(\forall [u'] \in \mathscr{V}_\omega)\ ([u'] \in_{\mathscr{V}_\omega} can(/\!\backslash u) \ \Leftrightarrow \ (\forall [v] \in_{\mathscr{V}_\omega} [u])\ [u'] \in_{\mathscr{V}_\omega} [v])$.

(Note: You are not required to prove that `HF-SETS-EXT` *satisifies executability conditions (1)–(5): you can assume those conditions in your proof).*

Exercise 24 *Define functions* `vtimes`, `vexp`, *and* `vfactorial` *to perform multiplication, exponentiation, and factorial of von Neumann naturals. Define similar functions* `Ztimes`, `Zexp` *and* `Zfactorial` *for Zermelo naturals.*

Exercise 25 *Define a sort* `Pair` *of sets that are pairs. Likewise, define a sort* `OPair` *of sets that are ordered pairs, together with operations* `p1`*, resp.,* `p2`*, that extract the first, resp., second component of an ordered pair. (Hint: use memberships to carve out the appropriate sorts).*

Exercise 26 *Define a sort* `Graph` *of sets that are finite directed graphs. Define then a function* `TC` *that computes the transitive closure of a graph. (Hints: take advantage of the sort* `OPair` *defined in Exercise 25 and use memberships. You may find it useful to exploit Maude's* `otherwise` *feature (see [10] 4.5.4).*

Exercise 27 *(Generation* ex nihilo*). It follows trivially from inspecting the signature of* `HF-SETS` *that all terms of sort* `Set` *are generated from* `0` *by iterating the constructors* `{_}` *and* `_,_`*, that is, essentailly by pairing. But can we instead generate any HF-set from* `0` *by iterating the defined operations* `_U_` *and* `P`*? Consider the following functional module:*

```
fmod HF-SETS-EXT1 is protecting HF-SETS-EXT .
  ops pow gen : ZNat -> Set .
  var J : ZNat .
  eq pow(0) = 0 .
  eq pow({J}) = P(pow(J)) .
  eq gen(0) = 0 .
  eq gen({J}) = gen(J) U pow({J}) .
endfm
```

Prove that `gen` *generates all of* $\mathscr{V}_\omega$*, that is, if we define* $\mathbb{N}_Z = Can_{\texttt{HF-SETS-EXT,ZNat}}$*, then we have:*

$$(\forall [u] \in \mathscr{V}_\omega)(\exists [j] \in \mathbb{N}_Z)\ ([u] \in_{\mathscr{V}_\omega} can(\texttt{gen}(j)).$$

5.6 HF-Sets with Atoms

Exercise 27 makes clear that any HF-set can be generated from the empty set by repeated aplication of powerset and union operations. We shall see in §19 that this holds true for sets in general. While this is conceptually pleasing and achieves a useful economy of thought, it does not come for free: one has to pay a high representational cost, since everything under the sun has to be *represented* as a set built up from $\emptyset$. In mathematical practice, of course, nobody does that: we use sets such as $\{17, 213, 7, 4/7\}$, $\{a, b, c, d\}$, $\{f(a), g(a, x), h(z)\}$, $\{\pi, e, \sqrt[2]{2}\}$, and so on, depending on the particular domain of interest we are working on, without bothering to explicitly represent such elements as sets, even though in principle we could. All of set theory can be developed for sets that may contain *atoms* (also called *urelements*), which can be *elements* of sets, but are not themselves sets. The only drawback of this alternative formulation is that it makes the theoretical development slightly more complicated.

All this suggests lowering the representational cost of set theory for our computational model `HF-SETS` of HF-sets by allowing it to include any atoms we may wish to have. This can also drastically lower its *computational cost*, since we can choose the most efficient data representation for each particular kind of data, instead of adopting the one-size-fits-all approach of representing everything as a set built up from $\emptyset$. For example, the Zermelo representation of the natural numbers is much more eficient thant the von Neumann representation, since we have to perform logical inference (as opposed to parsing) to just type a set as a von Neumann natural number. But both are much less efficient than an arbitrary precision natural number representation as the one used in Maude's predefined `NAT` module. Atoms can also give us more flexibility for deploying our set theory exactly where we needed it, as opposet to just everywhere. For example we may want to have the freedom to reason about finite *sets* of numbers, without having to represent numbers also as sets, which can make our reasoning simpler.

In summary, we would like to generalize `HF-SETS` so as to make it possible to include different kinds of data as atoms in HF-sets. How can we do this? We should think of such a module as a *parameterized* or "polymorphic" module, entirely analogous to a parameterized module for, say, lists, which can be instantiatied to build lists of booleans, or list of reals, or lists of character strings, depending on the kind of data we

choose for the list elements. In Maude this can be done by means of a *parameterized functional module*, which is module of the form $\texttt{fmod}(\Sigma, E \cup A)\{X :: (\Sigma', E')\}\texttt{endfm}$, where (Σ', E') is an equational theory called the *parameter theory*, and X is a *name* that allows us to refer to it. The role of the parameter theory is to specify the *semantic requirements* that a legal instance of the module should satisfy, so we should think of (Σ', E') as a *semantic interface*. For example, the parameter theory of a sorting module should be the theory of totally ordered sets, since otherwise our parameterized sorting program may produce strange answers. Mathematically, what the notation $\texttt{fmod}(\Sigma, E \cup A)\{X :: (\Sigma', E')\}\texttt{endfm}$ describes is just a *theory inclusion* $(\Sigma', E') \subseteq (\Sigma, E \cup A)$ of the parameter theory into the body's theory, that is, the sorts, operations, and axioms of (Σ', E') are all included in $(\Sigma, E \cup A)$. However, although we want the body itself, that is, the fragment $(\Sigma, E \cup A) - (\Sigma', E')$, to satisfy a parametric version of the executabilty requirements (1)–(5), the parameter theory (Σ', E') is only used to specify semantic requirements, not for computation, and *need not be executable* at all. So, we are looking for a parameterized module of the form `HF-SETS{X :: ATOM}`, with `ATOM` its parameter theory. But what semantic requirements should `ATOM` specify? Certainly, we would like to choose any set of atoms as an instance, but we need more than just a set. Consider equations `[3]` and `[4]` in `HF-SETS`, which define set membership in terms of set equality. What should those equations be like when instead of testing for membership of a set `S` we test for membership of an atom `A`? Unless atoms come with an *equality predicate*, we will not know what to do. Here is our desired module.

```
fth ATOM is protecting BOOL-OPS .
  sort Atom .
  op equal : Atom Atom -> Bool .
  vars A B : Atom .
  eq equal(A,A) = true .
  ceq A = B if equal(A,B) = true .
endfth

fmod HF-SETS{X :: ATOM} is
  sorts Elt{X} Magma{X} Set{X} .
  subsorts Set{X} < Elt{X} <  Magma{X} .
  op [_] : X$Atom -> Elt{X} [ctor] .
  op _,_ : Magma{X} Magma{X} -> Magma{X} [ctor assoc comm] .
  op {_} : Magma{X} -> Set{X} [ctor] .
  op 0 : -> Set{X} [ctor] .

  vars M M' : Magma{X} .  vars S S' T : Set{X} .
  vars A B : X$Atom .     var E : Elt{X} .

  eq [1]: M, M = M .

  op _in_ : Magma{X} Set{X} -> Bool .
  eq [2]: M in 0 = false .
  eq [3]: S in {S'} = S ~ S' .
  eq [4]: S in {[A]} = false .
  eq [5]: S in {S',M} = (S ~ S') or (S in {M}) .
  eq [6]: S in {[A],M} = S in {M} .
  eq [7]: [A] in {S} = false .
  eq [8]: [A] in {[B]} = equal(A,B) .
  eq [9]: [A] in {S,M} = [A] in {M} .
  eq [10]: [A] in {[B],M} = (equal(A,B)) or ([A] in {M}) .
  eq [11]: E, M in {M'} = (E in {M'}) and (M in {M'}) .

  op _~_ : Set{X} Set{X} -> Bool .
  eq [12]: S ~ S' = (S <= S') and (S' <= S) .

  op _<=_ : Set{X} Set{X} -> Bool .
  eq [13]: 0 <= S = true .
  eq [14]: {M} <= S = M in S .

  op _U_ : Set{X} Set{X} -> Set{X} [assoc comm] .
  eq [15]: S U 0 = S .
  eq [16]: {M} U {M'} = {M,M'} .

  op P : Set{X} -> Set{X} .
  eq [17]: P(0) = {0} .
  eq [18]: P({E}) = {0,{E}} .
```

```
    eq [19]: P({E,M}) = P({M}) U augment(P({M}),E) .

    op augment : Set{X} Elt{X} -> Set{X} .
    eq [20]: augment(0,E) = 0 .
    eq [21]: augment({S},E) = {{E} U S} .
    eq [22]: augment({[A]},E) = {{E,[A]}} .
    eq [23]: augment({M,M'},E) = augment({M},E) U augment({M'},E) .
  endfm
```

Maude parameter theories are specified with syntax `fth`(Σ', E')`endfth`. The `protecting BOOL-OPS` declaration specifies not only that the functional module `BOOL-OPS` is imported by `ATOM`, but also that its *standard* interpretation –with only two elements `true` and `false`, and with `true` $\neq$ `false`– is intended. The two equations in `ATOM` then force `equal` to be an equality predicate, so that `equal(A,B) = true` iff `A = B`, and `equal(A,B) = false` iff `A` $\neq$ `B`. The parameterized module `HF-SETS{X :: ATOM}` generalizes `HF-SETS` in a very natural way. All sorts now become parametric, and a new sort `Elt{X}` is added, since now a set element can be *either* a set, *or* an atom. The constructor `[_]` inserts each element of the parameter sort `X$Atom` into the sort `Elt{X}`. This is safer than giving a subsort declaration `X$Atom < Elt{X}`, because then if we were to choose numbers as atoms we would encounter both syntactic ambiguities and semantic contradictions : is `0` the zero number or the empty set? and is `s(0)` the number 1 in Peano notation or the set `{0}`? Also, should `s(0) in {s(0)}` evaluate to `true` or to `false` ? The constructor `[_]` acts as a *cordon sanitaire* between the two territories of atoms and of sets to avoid these problems. All equations are almost identical to those in `HF-SETS`, but we now need a few more equations, particularly for `_in_`, since there are more cases to cover depending on whether an element is an atom or a set.

Of course, since `HF-SETS{X :: ATOM}` is parametric, we need to choose how to *instantiate* it with a particular choice of atoms. In Maude this is done with a *view*, which is just a *theory intepretation*, that is, a mapping $V : (\Sigma', E') \longrightarrow (\Sigma'', E'')$ from the parameter theory (Σ', E') in question (in our case the theory `ATOM`) to the theory (Σ'', E'') of the *target module* where we want to interpret (Σ', E'). Specifically, V should map sorts to sorts, and operation symbols to operation symbols. Furthermore, V should be *semantics-preserving*. This means that the translated axioms $V(E')$ should be theorems[8] in the target theory (Σ'', E''). That is, V allows us to "view" (Σ'', E'') as an instance of (Σ', E'). For example, if we want our atoms to be natural numbers, we can define the view:

```
view Nat-eq from ATOM to NAT is
  sort Atom to Nat .
  op equal to _==_ .
endv
```

from `ATOM` to Maude's predefined `NAT` module, by maping the sort `Atom` to the sort `Nat`, and the operator `equal` to the buil-in equality predicate `_==_` on natural numbers. We can then instantiate `HF-SETS{X :: ATOM}` with this view to get HF-sets with numbers as atoms as follows:

```
fmod HF-SETS-NAT is protecting HF-SETS{Nat-eq} . endfm
```

Let us perform some reductions in `HF-SETS-NAT`:

```
Maude> red {[1]} in {[0],0,{[1]}} .
result Bool: true

Maude> red {[1],0,[1],{[1],[2]}} ~ {[1],0,{[1],[2]}} .
result Bool: true

Maude> red {[1],0,[1]} <= {[1],0,{[1],[2]}} .
result Bool: true

Maude> red {[1],0,{{[2]}}} U {[1],{[2],[3]}} .
result Set{Nat-eq}: {0,[1],{{[2]}},{[2],[3]}}

Maude> red P({[1],[2]}) .
result Set{Nat-eq}: {0,{[1]},{[2]},{[1],[2]}}
```

[8] Maude does not check that the axioms $V(E')$ are theorems in (Σ'', E''), but Maude's ITP tool can be used for this purpose.

We can likewise choose quoted identifiers as our atoms by defining the following view and module instantiation:

```
view Qid-eq from ATOM to QID is
 sort Atom to Qid .
 op equal to _==_ .
endv

fmod HF-SETS-QID is protecting HF-SETS{Qid-eq} . endfm
```

Here are some sample reductions for `HF-SETS-QID`:

```
Maude> red {['a]},0 in {0,{['a],['b]},{['a]},['d]} .
result Bool: true

Maude> red {{['a]},0} ~ {0,{['a],['b]},{['a]},['d]} .
result Bool: false

Maude> red {{['a]},0} <= {0,{['a],['b]},{['a]},['d]} .
result Bool: true

Maude> red {{['a]},0,['e]} U  {0,{['a],['b]},{['a]},['d]} .
result Set{Qid-eq}: {0,['d],['e],{['a]},{['a],['b]}}

Maude> red P({['a],['b],['c]}) .
result Set{Qid-eq}: {0,{['a]},{['b]},{['c]},{['a],['b]},{['a],['c]},
                     {['b],['c]},{['a],['b],['c]}}
```

Chapter 6

Relations, Functions, and Function Sets

Relations and functions are pervasive, not just in mathematics but in natural language and therefore in ordinary life: we cannot open our mouth for very long without invoking a relation or a function. When someone says, "my mother is Judy Tuesday," that person is invoking a well-known function that assigns to each non-biologically-engineered human being his or her natural mother. Likewise, when someone says "our four grandparents came for dinner," he/she is invoking a well-known relation of being a grandparent, which holds between two human beings x and z iff z is a child of some y who, in turn, is a child of x. One of the key ways in which set theory is an excellent mathematical modeling language is precisely by how easily and naturally relations and functions can be represented as *sets*. Furthermore, set theory makes clear how relations and functions can be *composed*, giving rise to new relations and functions.

6.1 Relations and Functions

How does set theory model a relation? Typically there will be two sets of entities, say A and B, so that the relation "relates" some elements of A to some elements of B. In some cases, of course, we may have $A = B$. For example, in the "greater than" relation, $>$, between natural numbers, we have $A = B = \mathbb{N}$; but in general A and B may be different sets.

So, what is a *relation*? The answer is quite obvious: *a relation is exactly a set of ordered pairs in some cartesian product*. That is, a *relation* is exactly a *subset* of a cartesian product $A \times B$, that is, an element of the powerset $\mathcal{P}(A \times B)$ for some sets A and B. We typically use capital letters like R, G, H, etc., to denote relations. By convention we write $R : A \Longrightarrow B$ as a useful, shorthand notation for $R \in \mathcal{P}(A \times B)$, and say that "$R$ is a relation from A to B," or "R is a relation whose *domain*[1] is A and whose *codomain* (or *range*) is B." Sometimes, instead of writing $(a, b) \in R$ to indicate that a pair (a, b) is in the relation R, we can use the more intuitive infix notation $a\,R\,b$. This infix notation is quite common. For example, we write $7 > 5$, to state that 7 is greater than 5, instead of the more awkward (but equivalent) notation $(7, 5) \in >$.

Note that given a relation $R \subseteq A \times B$ we can define its *inverse* relation $R^{-1} \subseteq B \times A$ as the set $R^{-1} = \{(y, x) \in B \times A \mid (x, y) \in R\}$. The idea of an inverse relation is of course pervasive in natural language: "grandchild" is the inverse relation of "grandparent," and "child" is the inverse relation of "parent." It follows immediately from this definition that for any relation R we have, $(R^{-1})^{-1} = R$.

What is a function? Again, typically a function f will map an element x of a set A to corresponding elements $f(x)$ of a set B. So the obvious answer is that a function is a *special kind of relation*. Which kind? Well, f is a relation that must be *defined* for every element $x \in A$, and must relate each element x of A to a *unique* element $f(x)$ of B. This brings us to the following question: we know that, given sets A and B, the *set of all relations* from A to B is precisely the powerset $\mathcal{P}(A \times B)$. But what is the *set of all functions* from A to B? Obviously a *subset* of $\mathcal{P}(A \times B)$, which we denote $[A{\to}B]$ and call the *function set from* A to B. Can we define $[A{\to}B]$ precisely? Yes, of course:

$$[A{\to}B] = \{f \in \mathcal{P}(A \times B) \mid (\forall a \in A)(\exists! b \in B)\ (a, b) \in f\}.$$

[1] This does not necessarily imply that R is "defined" for all $a \in A$, that is, we do not require that $(\forall a \in A)(\exists b \in B)(a, b) \in R$.

We can introduce some useful notation for functions. Typically (but not always) we will use lower case letters like f, g, h, and so on, to denote functions. By convention, we write $f : A \longrightarrow B$ as a shorthand notation for $f \in [A \to B]$. We then read $f : A \longrightarrow B$ as "f is a function from A to B," or "f is a function whose *domain* is A, and whose *codomain* (also called *range*) is B." Also, if $f : A \longrightarrow B$ and $a \in A$, the unique $b \in B$ such that $a\, f\, b$ is denoted $f(a)$. This is of course the standard notation for function application, well-known in algebra and calculus, where we write expressions such as $factorial(7)$, $2 + 2$ (which in the above prefix notation would be rendered $+(2, 2)$), $sin(\pi)$, and so on.

Exercise 28 *Let A and B be finite sets. If A has n elements and B has m elements, how many relations are there from A to B? And how many functions are there from A to B? Give a detailed proof of your answers.*

6.2 Formula, Assignment, and Lambda Notations

This is all very well, but how can we *specify* in a precise way a given relation or function we want to use? If set theory is such a good modeling language, it should provide a way to unambiguously specify whatever concrete relation or function we want to define. The fact that we know that the set of all relations from A to B is the powerset $\mathcal{P}(A \times B)$, and that the set of all functions from A to B is the set $[A \to B]$ is well and good; but that does not tell us anything about how to specify any concrete relation $R \in \mathcal{P}(A \times B)$, or any concrete function $f \in [A \to B]$.

Essentially, there are two ways to go: the hard way, and the easy way. The hard way only applies under some finiteness assumptions. If A and B are *finite* sets, then we know that $A \times B$ is also a finite set; and then any $R \in \mathcal{P}(A \times B)$, that is, any subset $R \subset A \times B$ is also finite. So we can just *list* the elements making up the relation R, say, $R = \{(a_1, b_1), \ldots, (a_n, b_n)\}$. This is exactly the way a finite relation is stored in a *relational data base*, namely, as the set of *tuples* in the relation.[2] For example, a university database may store the relationship between students and the courses each student takes in a given semester in exactly this way: as a finite set of pairs. In reality, we need not require A and B to be finite sets in order for this explicit listing of R to be possible: it is enough to require that R itself is a finite set. Of course, for finite functions f we can do just the same: we can specify f as the set of its pairs $f = \{(a_1, b_1), \ldots, (a_n, b_n)\}$. However, since a function $f : A \longrightarrow B$ must be defined for all $a \in A$, it follows trivially that f is finite iff A is finite.

So long for the hard way. How about the easy way? The easy way is to represent relations and functions *symbolically*, or, as philosophers like to say, *intensionally*. That is, by a piece of language, which is always finite, even though what it describes (its "extension") may be infinite. Of course, for this way of specifying relations and functions to work, our language must be completely precise, but we are in very good shape in this regard. Isn't set theory a precise formal language for *all* of mathematics? So we can just *agree* that a precise linguistic description of a relation R is just a *formula* φ in set theory with exactly two free variables, x and y. Then, given domain and codomain sets A and B, this formula φ unambiguously defines a relation $R \subseteq A \times B$, namely, the relation

$$R = \{(x, y) \in A \times B \mid \varphi\}.$$

For example, the greater than relation, $>$, on the von Neumann natural numbers can be specified by the set theory formula $y \in x$, since we have,

$$> = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid y \in x\}.$$

Similarly, the square root relation on real numbers is specified by the formula[3] $y^2 = x$, defining for us the set of pairs $SQRT = \{(x, y) \in \mathbb{R}^2 \mid y^2 = x\}$, which geometrically is a parabola. Note that what we are exploiting here is the axiom scheme (*Sep*) of separation, which endows set theory with enormous expressive power to use its own "metalanguage" in order to specify sets. Let us call this syntactic way of defining relations the *formula notation*.

How can we likewise define functions symbolically? Since functions are a special kind of relation, we can always use the above formula notation to specify functions, but this may not be a very good idea.

[2] The fact that the tuples or "records," may have more than two elements in them does not really matter for our modeling purposes, since we can view, say, a triple (a, b, c) as a pair $(a, (b, c))$, and, similarly, and n-tuple $(a_1, \ldots, a_n)$ as a pair $(a_1, (a_2, \ldots, a_n))$.

[3] Admittedly, the expression y^2 for squaring a real number does not belong to our basic set theory language; however, as explained in what follows, such a language can be extended so that it *does* belong to an extended set theory language. Such language extensions are the natural result of modeling all of mathematics within set theory, and are also very useful for set theory itself.

Why not? Because just from looking at a formula $\varphi(x, y)$ we may not have an obvious way to know that for each x there will always be a unique y such that $\varphi(x, y)$ holds, and we need this to be the case if $\varphi(x, y)$ is going to define a function. A better way is to use a set-theoretic *term* or *expression* to define a function. For example, we can use the set-theoretic expression $x \cup \{x\}$ to define the successor function $s : \mathbb{N} \longrightarrow \mathbb{N}$ on the von Neumann natural numbers. We can do this using two different notations, called, respectively, the *assignment* notation and the *lambda* notation. In the assignment notation we write, for example, $s : \mathbb{N} \longrightarrow \mathbb{N} : x \mapsto x \cup \{x\}$ to unambiguously define the successor function. More generally, if $t(x)$ is a set-theoretic term or expression having a single variable x (or having no variable at all), then we can write $f : A \longrightarrow B : x \mapsto t(x)$ to define the function f. In which sense is f defined? That is, how is f specified as a *set*? Of course it is specified as the set

$$f = \{(x, y) \in A \times B \mid y = t(x)\}.$$

Ok, but how do we *know* that such a set is a function? Well, we do not quite know. There is a possibility that the whole thing is nonsense and we have *not* defined a function. Certainly the above set is a relation from A to B. Furthermore, we cannot have $(a, b), (a, b') \in f$ with $b \neq b'$, since $t(x)$ is a term, and this forces $b = t(a) = b'$. The rub comes from the fact that we could have $a \in A$ but $t(a) \notin B$. In such a case f will not be defined for all $a \in A$, which it must be in order for f to be a function. In summary, the assignment notation, if used senselessly, may not define a function, but only what is called a *partial* function. In order for this notation to really define a function from A to B, we must furthermore check that for each $a \in A$ the element $t(a)$ belongs to the set B. An alternative, quite compact variant of the assignment notation is the notation $A \ni x \mapsto t(x) \in B$.

What about the *lambda notation*? This notation is also based on the idea of symbolically specifying a function by means of a term $t(x)$. It is just a syntactic variant of the assignment notation. Instead of writing $x \mapsto t(x)$ we write $\lambda x.\ t(x)$. To make explicit the domain A and codomain B of the function so defined we should write $\lambda x \in A.\ t(x) \in B$. Again, this could fail to define a function if for some $a \in A$ we have $t(a) \notin B$, so we have to check that for each $a \in A$ we indeed have $t(a) \in B$. For example, assuming that we have already defined the addition function $+$ on the natural numbers, we could define the function $double : \mathbb{N} \longrightarrow \mathbb{N}$ by the defining equality $double = \lambda x \in \mathbb{N}.\ x + x \in \mathbb{N}$. A good point about the lambda notation $\lambda x.\ t(x)$ is that the λ symbol makes explicit that it is used as a "binder" or "quantifier" that binds its argument variable x. This means that the particular choice of x as a variable is immaterial. We could instead write $\lambda y.\ t(y)$ and this would define the same function. Of course, this also happens for the assigment notation: we can write $A \ni y \mapsto t(y) \in B$ instead of $A \ni x \mapsto t(x) \in B$, since both notations will define the same function.

Both the assignment and the lambda notations have easy generalizations to notations for defining functions of several arguments, that is, functions of the form $f : A_1 \times \ldots \times A_n \longrightarrow B$. Instead of choosing a term $t(x)$ with (at most) a single variable, we now choose a term $t(x_1, \ldots, x_n)$ with (at most) n variables and write $f : A_1 \times \ldots \times A_n \longrightarrow B : (x_1, \ldots, x_n) \mapsto t(x_1, \ldots, x_n)$, or $A_1 \times \ldots \times A_n \ni (x_1, \ldots, x_n) \mapsto t(x_1, \ldots, x_n) \in B$ in assignment notation; or $\lambda(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n.\ t(x_1, \ldots, x_n) \in B$ in lambda notation. For example, we can define the average function on a pair of real numbers by the assignment notation: $av : \mathbb{R}^2 \longrightarrow \mathbb{R} : (x, y) \mapsto (x + y)/2$. Of course, the function f defined by the assignment notation $A_1 \times \ldots \times A_n \ni (x_1, \ldots, x_n) \mapsto t(x_1, \ldots, x_n) \in B$, or the lambda notation $\lambda(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n.\ t(x_1, \ldots, x_n) \in B$, is:

$$f = \{((x_1, \ldots, x_n), y) \in (A_1 \times \ldots \times A_n) \times B \mid y = t(x_1, \ldots, x_n)\}.$$

Perhaps a nagging doubt that should be assuaged is what terms, say, $t(x)$ or $t(x_1, \ldots, x_n)$, are we allowed to use in our set theory language. After all, in the original formal language of set theory presented in §2 the only terms allowed were variables! This certainly will not carry us very far in defining functions. The answer to this question is that we are free to use any terms or expressions available to us in any *definitional extension of the set theory language*. The idea of a *definitional extension* of a first-order language is very simple: we can always add new, auxiliary notation, provided this new notation is precisely defined *in terms of the previous notation*. We have been using this idea already quite a lot when introducing new auxiliary symbols like $\varnothing$, $\subseteq$, $\bigcup$, $\bigcap$, $\boxplus$, $\otimes$, $\oplus$, $\times$, or $\mathcal{P}$, in our set theory language. For example, we defined the containment relation $\subseteq$ in terms of the basic notation of set theory —which only uses the $\in$ binary relation symbol and the built-in equality symbol— by means of the defining equivalence $x \subseteq y \iff (\forall z)(z \in x \Rightarrow z \in y)$. Similarly, the term $x \cup \{x\}$ that we used in defining the successor function on von Neumann natural

numbers became perfectly defined after singleton sets were formally defined by means of the (*Pair*) axiom and the union operation was formally defined by means of the (*Union*) axiom.

More precisely, given any formula φ in our language whose free variables are exactly $x_1, \ldots, x_n$, we can introduce a new predicate symbol, say P, of n arguments as an abbreviation for it, provided we add the axiom $P(x_1, \ldots, x_n) \Leftrightarrow \varphi$, which uniquely defines the meaning of P in terms of φ. For example, the predicate $\subseteq$ is defined in this way by the equivalence $x \subseteq y \Leftrightarrow (\forall z \in x)\ z \in y$. Similarly, if we have a formula φ in our language whose free variables are exactly $x_1, \ldots, x_n, y$, and we can *prove* that the formula $(\forall x_1, \ldots, x_n)(\exists! y)\, \varphi$ is a theorem of set theory, then we can introduce in our language a new function symbol f of n arguments, and we can define the meaning of f by adding the axiom $f(x_1, \ldots, x_n) = y \Leftrightarrow \varphi$. For example, we have done exactly this to define $\{x_1, x_2\}$, $\bigcup x$, and $\mathcal{P}(x)$, using the uniquely existentially quantified variable y in, respectively, the (*Pair*), (*Union*), and (*Pow*) axioms. Of course, this language extension process can be iterated: we can first define some new function and predicate symbols this way, and can later introduce other new symbols by defining them in terms of the formulas in the previous language extension. For example, the sucessor function can be defined in terms of binary union $\_ \cup \_$ and the pairing operation $\{\_, \_\}$ by means of the term $x \cup \{x\}$, which abbreviates the term $x \cup \{x, x\}$. Finally, by repeatedly replacing each predicate or function symbol by its corresponding definition, we can always "define away" everything into their simplest possible formulation in the basic language of set theory presented in § 2.

Exercise 29 *Define away the formula $y = s(x)$, where s is the successor function, into its equivalent formula in the basic language of set theory, which uses only variables and the $\in$ and $=$ predicates.*

Exercise 30 *(Leibniz Equality). The great 17th Century mathematician, logician, and philosopher Gottfried Wilhelm Leibniz proposed the following criterion for the equality of any two entities a and b:*

(L) a equals b iff for any property $P(x)$, $P(a)$ hods iff $P(b)$ holds.

That is, two things are equal iff they satisfy exactly the same properties. This is in a sense a behavioral *notion of equality, since we can think of each property P as a "test" to which a and b are submitted to try to tell them apart. If they behave the same for all possible tests, then, Lebniz says, they must be equal.*

Can we capture Leibniz's idea in the language of set theory? Since set theory is a universal language for mathematics, the general notion of a property $P(x)$ can be naturally identified (at least for properties of mathematical *entities), with that of a set theory formula $\varphi(x)$ with a single free variable x. And since in mathematics* any *entity can be modeled as a set, we can then rephrase Leibniz equality by saying that, given any two sets a and b,*

(L') a equals b iff for any set theory formula $\varphi(x)$, $\varphi(a)$ hods iff $\varphi(b)$ holds.

However, this rephrasing (L') is not *a set theory formula; it is a* metalevel statement *quantifiying over all set theory formulas with a single free variable. Can we find a* set theory formula *that can replace (L')?*

- *Prove that for any set theory formula $\varphi(x)$ with a single free varible x and for any two sets a and b, there is a set $A^{\varphi}_{a,b}$ such that $\varphi(a) \Leftrightarrow \varphi(b)$ holds iff $a \in A^{\varphi}_{a,b} \Leftrightarrow b \in A^{\varphi}_{a,b}$ holds.*

Since, given any two sets a, b, the intensional tests $\varphi(x)$ can be replaced by the extensional tests $x \in A^{\varphi}_{a,b}$, this sugests the even more general idea of replacing the intensional tests $\varphi(x)$ by the extensional tests $x \in A$, where A can be any *set. In this way we arrive at the following extensional formulation of Leibniz equality as the set theory formula:*

(L'') $a = b \Leftrightarrow (\forall A)\ (a \in A \Leftrightarrow b \in A)$.

Note that this is a definitional extension*, defining away the equality predicate $=$ (which we had assumed as buit-in in our set theory language) in terms of the membership predicate $\in$. Thechnically, what this means is that set theory can be defined in a first order language* without *equality, having just the membership predicate $\in$. But is (L'') true?*

- *Prove that the fomula $(\forall a, b)\ (a = b \Leftrightarrow (\forall A)\ (a \in A \Leftrightarrow b \in A))$ is a theorem of set theory.*

6.3 Images

Given a relation $R \subseteq A \times B$, we can consider the *image* under R of any subset $A' \in \mathcal{P}(A)$, that is, those elements of B related by R to some element in A'. The obvious definition is then,

$$R[A'] = \{b \in B \mid (\exists a \in A')\ (a, b) \in R\}.$$

For example, for $SQRT$ the square root relation on the set $\mathbb{R}$ of real numbers we have $SQRT[\{4\}] = \{2, -2\}$, $SQRT[\{-1\}] = \varnothing$, and $SQRT[\mathbb{R}] = \mathbb{R}$.

Of course, given $B' \subseteq B$, its *inverse image* under R,

$$R^{-1}[B'] = \{a \in A \mid (\exists b \in B')\ (a, b) \in R\},$$

is exactly the *image* of B' under the inverse relation R^{-1}. For example, since the inverse relation $SQRT^{-1}$ is the *square* function $\mathbb{R} \ni x \mapsto x^2 \in \mathbb{R}$, we have $SQRT^{-1}[\{-1\}] = \{1\}$, and $SQRT^{-1}[\mathbb{R}] = \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ denotes the set of positive (or zero) real numbers.

Given a relation $R \subseteq A \times B$, we call the set $R[A] \subseteq B$ the *image* of R. For example, the image of the square root relation $SQRT$ is $SQRT[\mathbb{R}] = \mathbb{R}$; and the image of the *square* function is $square[\mathbb{R}] = \mathbb{R}_{\geq 0}$.

Note that for any relation $R \subseteq A \times B$, the assignment $\mathcal{P}(A) \ni A' \mapsto R[A'] \in \mathcal{P}(B)$ defines a function $R[\_] : \mathcal{P}(A) \longrightarrow \mathcal{P}(B)$. In particular, for the inverse relation R^{-1} we have a function $R^{-1}[\_] : \mathcal{P}(B) \longrightarrow \mathcal{P}(A)$.

Note, finally, that when $f \subseteq A \times B$ is not just a relation but in fact a function $f : A \longrightarrow B$, then for each $a \in A$ we have *two different notations*, giving us two different results. On the one hand $f(a)$ gives us the unique $b \in B$ such that $a\, f\, b$, while on the other hand $f[\{a\}]$ gives us the *singleton* set whose only element is $f(a)$, that is, $f[\{a\}] = \{f(a)\}$.

Exercise 31 *Prove the following:*

1. *For any sets A, B, and C, if $f \in [A{\to}B]$ and $B \subseteq C$, then $f \in [A{\to}C]$, that is, $[A{\to}B] \subseteq [A{\to}C]$, so that we can always enlarge the* codomain *of a function to a bigger one.*
2. *Given $f \in [A{\to}B]$, prove that for any set C such that $f[A] \subseteq C \subseteq B$ we have $f \in [A{\to}C]$; that is, we can* restrict the codomain *of a function f to a smaller one C,* provided $f[A] \subseteq C$.
3. *If $f \in [A{\to}B]$ and $A \subset A'$ then $f \notin [A'{\to}B]$, that is, we cannot strictly enlarge the domain of a function f and still have f be a function. Come up with a precise set-theoretic definition of* partial function*, so that, under this definition, if $f \in [A{\to}B]$ and $A \subseteq A'$ then f is a partial function from A′ to B. Summarizing (1)–(3),* the domain of a function $f \in [A{\to}B]$ is fixed*, but* its codomain can always be enlarged*; and can also be restricted, provided the restricted codomain contains $f[A]$.*
4. *Call a relation R from A to B* total[4] *iff for each $a \in A$ we have $R[\{a\}] \neq \emptyset$. Show that any function from A to B is a total relation. Show also that if $f \in [A{\to}B]$ and $A \subset A'$, then f, as a relation from A′ to B, is not total (so that calling f a partial function from A′ to B makes sense). Come up with the most natural and economic possible way of extending any relation R from a set A to itself, that is, $R \in \mathcal{P}(A^2)$, to a total relation $R^\bullet$, so that if R is already total, then $R = R^\bullet$.*
5. *For any sets A, B, C, and D, if $R \in \mathcal{P}(A \times B)$, $A \subseteq C$, and $B \subseteq D$, then $R \in \mathcal{P}(C \times D)$, that is, we can always enlarge both the* domain *and the* codomain *of a relation. However, show that if $R \in \mathcal{P}(A \times B)$ is total from A to B and $A \subset C$, and $B \subseteq D$, then, R, as a relation from C to D, is* never *total.*

Exercise 32 *Given a function $f \in [A{\to}B]$ and given a subset $A' \subseteq A$ we can define the* restriction *$f{\restriction}_{A'}$ of f to A′ as the set*

$$f{\restriction}_{A'} = \{(a, b) \in f \mid a \in A'\}.$$

Prove that the assignment $f \mapsto f{\restriction}_{A'}$ then defines a function $\_{\restriction}_{A'} : [A{\to}B] \longrightarrow [A'{\to}B]$. Show, using the inclusion $\mathbb{N} \subset \mathbb{Z}$ of the natural numbers into the set $\mathbb{Z}$ of integers, as well as Exercise 31-(1)-(2), that the addition and multiplication functions on natural numbers are restrictions of the addition and multiplication functions on integers in exactly this sense.

Similarly, given a relation $R \in \mathcal{P}(A \times B)$ and given a subset $A' \subseteq A$ we can define the restriction *$R{\restriction}_{A'}$ of R to A′ as the set*

$$R{\restriction}_{A'} = \{(a, b) \in R \mid a \in A'\}.$$

Prove that the assignment $R \mapsto R{\restriction}_{A'}$ then defines a function $\_{\restriction}_{A'} : \mathcal{P}(A \times B) \longrightarrow \mathcal{P}(A' \times B)$.

[4] Please note that this use of the word "total" means "totally defined" and is opposed to "partial," in the sense of "partially defined," that is, a relation $R : A \Longrightarrow B$ is total iff it is defined for every $a \in A$, and should be called partial otherwise. All functions are total relations in this sense; and we call a relation $f : A \Longrightarrow B$ a *partial function* iff there is a subset $A' \subseteq A$ such that $f : A' \longrightarrow B$ is a (total) function. Note that this notion of total relation is *completely different* from another notion in which $R : A \Longrightarrow A$ is called "total" iff $(\forall x, y \in A)\ xRy \vee yRx$. This second sense of "total" is used in the notion of a *totally ordered set* with an order relation $\leq$ in Section 9.4. To make things even more confusing, an ordered set that is not total in this *second* sense is called a *partially ordered set* or *poset*; but here "partial" just means not (necessarily) total in this second and completely different sense. In what follows, the context will always make clear which of these two different senses of "total" or "partial" is meant.

6.4 Composing Relations and Functions

Given relations $F : A \Longrightarrow B$ and $G : B \Longrightarrow C$, their *composition* is the relation $F; G : A \Longrightarrow C$ defined as the set

$$F; G = \{(x, z) \in A \times C \mid (\exists y \in B)((x, y) \in F \ \wedge \ (y, z) \in G)\}.$$

Similarly, given functions $f : A \longrightarrow B$ and $g : B \longrightarrow C$, their *composition* is the relation $f; g : A \longrightarrow C$, which is trivial to check it is a function. The notation $F; G$ (resp., $f; g$) follows the *diagrammatic order*, so that F (resp., f) is the *first* relation (resp., function) from A to B, and G (resp., g) the *second* relation (resp., function) from B to C. Sometimes the composed relation $F; G$ (resp., composed function $f; g$) is denoted in *application order* as $G \circ F$ (resp., $g \circ f$). This is due to the convention of applying functions to an argument *on the left* of the argument, so to apply $f; g$ to $a \in A$ we have to apply f first to get $f(a)$, and then g to get $g(f(a))$. The notation $(g \circ f)(a) = g(f(a))$ then follows the application order for functions on the left, whereas the notation $f; g$ is easier to indicate that f is the *first* function applied, and g the *second*. We will allow *both notations*, but will favor the diagrammatic one.

Given any set A, the *identity function* on A is the function $id_A = \{(a, a) \mid a \in A\}$, or, in assignment notation, $id_A : A \longrightarrow A : a \mapsto a$. The following lemma is trivial to prove and is left as an exercise.

Lemma 4 *(Associativity and Identities for Relation and Function Composition)*

1. *Given relations $F : A \Longrightarrow B$, $G : B \Longrightarrow C$, and $H : C \Longrightarrow D$, their composition is* associative*, that is, we have the equality of relations $(F; G); H = F; (G; H)$.*

2. *Given functions $f : A \longrightarrow B$, $g : B \longrightarrow C$, and $h : C \longrightarrow D$, their composition is likewise* associative*, that is, we have the equality of functions $(f; g); h = f; (g; h)$.*

3. *Given a relation $F : A \Longrightarrow B$, we have the equalities $id_A; F = F$, and $F; id_B = F$.*

4. *Given a function $f : A \longrightarrow B$, we have the equalities $id_A; f = f$, and $f; id_B = f$.*

Closely related to the identity function id_A on a set A we more generally have inclusion functions. Given a subset inclusion $A' \subseteq A$, the *inclusion function* from A' to A is the function $j^A_{A'} = \{(a', a') \mid a' \in A'\}$, or, in assignment notation, $j^A_{A'} : A' \longrightarrow A : a' \mapsto a'$. That is, the function $j^A_{A'}$ is just the identity function $id_{A'}$ on A', except that we have extended its codomain from A' to A (see Exercise 31-(1)). To emphasize that an inclusion function identically includes the subset A' into A, we will use the notation $j^A_{A'} : A' \hookrightarrow A$. Note that inclusion functions are also closely connected with the notion of *restriction* $f{\restriction}_{A'}$ of a function $f : A \longrightarrow B$ or, more generally, restriction $F{\restriction}_{A'}$ of a relation $F : A \Longrightarrow B$ to a subset $A' \subseteq A$ of its domain defined in Exercise 32. Indeed, it is trivial to check that we have the equalities: $f{\restriction}_{A'} = j^A_{A'}; f$, and $F{\restriction}_{A'} = j^A_{A'}; F$.

We call a function $f : A \longrightarrow B$ *injective* iff $(\forall a, a' \in A)(f(a) = f(a') \Rightarrow a = a')$. Obviously, any inclusion function is injective. For another example of an injective function, consider multiplication by 2 (or by any nonzero number) on the natural numbers: $2 \cdot \_ : \mathbb{N} \longrightarrow \mathbb{N} : n \mapsto 2 \cdot n$. Similarly, addition by 2 (or by any natural number) $2 + \_ : \mathbb{N} \longrightarrow \mathbb{N} : n \mapsto 2 + n$ is an injective function. We use the notation $f : A \rightarrowtail B$ as a shorthand for "f is an injective function from A to B." Note that since an inclusion $j^A_{A'} : A' \hookrightarrow A$ is always injective, it could also be written $j^A_{A'} : A' \rightarrowtail A$.

We call a function $f : A \longrightarrow B$ *surjective* iff B is the image of f, that is, iff $f[A] = B$. For example, the absolute value function $|\_| : \mathbb{Z} \longrightarrow \mathbb{N}$, with $|n| = n$ if $n \in \mathbb{N}$, and $|\text{-}n| = n$ for negative numbers, is clearly surjective. Similarly, the projection to the horizontal plane $\pi : \mathbb{R}^3 \longrightarrow \mathbb{R}^2 : (x, y, z) \mapsto (x, y)$ is also surjective. We use the notation $f : A \twoheadrightarrow B$ as a shorthand for "f is a surjective function from A to B." Note that, taking into account Exercise 31, it is immediate to check that *any* function $f : A \longrightarrow B$ can be expressed in a unique way as a composition of its "surjective part" (the restriction of its codomain to $f[A]$), followed by the inclusion of $f[A]$ into B. That is, we always have $f = f; j^B_{f[A]}$, according to the composition

$$A \xrightarrow{f} f[A] \overset{j^B_{f[A]}}{\hookrightarrow} B.$$

Since any inclusion function is injective, the above composition $f = f; j^B_{f[A]}$ shows that any function can always be expressed as the composition of a surjective function followed by an injective function.

We call a function $f : A \longrightarrow B$ *bijective* iff it is both injective and surjective. Obviously, the identity function id_A is bijective. Similarly, the function mapping each point of the three dimensional space to its "mirror image" on the other side of the $x - z$ plane, $mirror : \mathbb{R}^3 \longrightarrow \mathbb{R}^3 : (x, y, z) \mapsto (x, -y, z)$ is also clearly bijective. We use the notation $f : A \xrightarrow{\cong} B$ as a shorthand for "f is a bijective function from A to B." We also use the notation $A \cong B$ as a shorthand for "there exists a bijective function from A to B." For example, $\mathcal{P}(\{0, 1, 2\}) \cong 8$.

Since any function $f : A \longrightarrow B$ is a special case of a relation, its inverse relation $f^{-1} : B \Longrightarrow A$ is always defined. However, in general f^{-1} is a *relation*, but not a function. We have already encountered instances of this phenomenon. For example, given the square function $square : \mathbb{R} \longrightarrow \mathbb{R} : x \mapsto x^2$, its inverse relation $square^{-1} : \mathbb{R} \Longrightarrow \mathbb{R}$ is precisely the square root relation $SQRT$, which is *not* a function. It is then interesting to ask: given a function $f : A \longrightarrow B$, when is its inverse relation f^{-1} a function? More generally, how is the inverse relation f^{-1} related to the injective, surjective, or bijective nature of a function?

Exercise 33 *Given a function $f : A \longrightarrow B$, prove the following:*

1. *We always have inclusions $f^{-1}; f \subseteq id_B$, and $id_A \subseteq f; f^{-1}$.*
2. *A relation $R : A \Longrightarrow B$ is a function iff $R^{-1}; R \subseteq id_B$, and $id_A \subseteq R; R^{-1}$.*
3. *f is surjective iff $f^{-1}; f = id_B$.*
4. *f is injective iff $f; f^{-1} = id_A$.*
5. *f is bijective iff the relation f^{-1} is a function $f^{-1} : B \longrightarrow A$.*
6. *f is bijective iff there exists a function $g : B \longrightarrow A$ such that $f; g = id_A$ and $g; f = id_B$. Furthermore, g satisfies these conditions iff f^{-1} is a function and $g = f^{-1}$.*

The last two characterizations in Exercise 33 clearly tell us that if we have two sets A and B such that there is a bijective function $f : A \xrightarrow{\cong} B$ between them, then in all relevant aspects these sets are "essentially the same," because we can put each element a of A in a "one-to-one" correspondence with a unique element of b, namely, $f(a)$, and conversely, each element b of B is put in a one-to-one correspondence with a unique element of A, namely, $f^{-1}(b)$. This means that f and f^{-1} act as *faithful encoding functions*, so that data from A can be faithfully encoded by data from B, and data from B can be faithfully decoded as data from A. It also means that the sets A and B have "the same size," something that we will explore in more depth in §§11, 15, and 18.

We can illustrate the way in which a bijection between two sets makes them "essentially the same" by explaining how the powerset construction $\mathcal{P}(A)$ and the function set construction $[A{\to}2]$ are intimately related in a bijective way. Given any set A, the sets $\mathcal{P}(A)$ and $[A{\to}2]$ are *essentially the same* in this precise, bijective sense. $\mathcal{P}(A)$ and $[A{\to}2]$ give us two alternative ways of dealing with a subset $B \subseteq A$. Viewed as an element $B \in \mathcal{P}(A)$, B is just a subset. But we can alternatively view B as a boolean-valued *predicate*, which is true for the elements of B, and false for the elements of $A - B$. That is, we can alternatively represent B as the function

$$\chi_B : A \longrightarrow 2 : a \mapsto \textbf{if } a \in B \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi}$$

where χ_B is called the *characteristic function* of the subset B. The sets $\mathcal{P}(A)$ and $[A{\to}2]$ are essentially the same, because the function $\chi_{\_} : \mathcal{P}(A) \longrightarrow [A{\to}2] : B \mapsto \chi_B$ is *bijective*, since its inverse is the function $(\_)^{-1}[\{1\}] : [A{\to}2] \longrightarrow \mathcal{P}(A) : f \mapsto f^{-1}[\{1\}]$.

We call a function $f : A \longrightarrow B$ a *left inverse* iff there is a function $g : B \longrightarrow A$ such that $f; g = id_A$. Similarly, we call $g : B \longrightarrow A$ a *right inverse* iff there is a function $f : A \longrightarrow B$ such that $f; g = id_A$. For example, composing

$$\mathbb{N} \overset{j^{\mathbb{Z}}_{\mathbb{N}}}{\hookrightarrow} \mathbb{Z} \overset{|\_|}{\twoheadrightarrow} \mathbb{N}$$

the inclusion of the natural numbers in the integers with the absolute value function, we obtain $j^{\mathbb{Z}}_{\mathbb{N}}; |\_| = id_{\mathbb{N}}$, and therefore $j^{\mathbb{Z}}_{\mathbb{N}}$ is a left inverse and $|\_|$ is a right inverse.

Exercise 34 *Prove the following:*

1. *If $f : A \longrightarrow B$ is a left inverse, then f is injective.*
2. *If $f : A \longrightarrow B$ is injective and $A \neq \emptyset$, then f is a left inverse.*
3. *If $f : A \longrightarrow B$ is a right inverse, then f is surjective.*

4. *$f : A \longrightarrow B$ is both a left and a right inverse iff f is bijective.*

Is every surjective function a right inverse? As we shall see, that depends on the set theory axioms that we assume. We shall revisit this question in §13.2.

Exercise 35 *(Epis and Monos). Call a function $f : A \longrightarrow B$* epi *iff for any set C and any two functions $g, h : B \longrightarrow C$, if $f; g = f; h$ then $g = h$. Dually,*[5] *call a function $f : A \longrightarrow B$* mono *iff for any set C and any two functions $g, h : C \longrightarrow A$, if $g; f = h; f$ then $g = h$. Prove the following:*

1. *$f : A \longrightarrow B$ is epi iff f is surjective.*
2. *$f : A \longrightarrow B$ is mono iff f is injective.*
3. *If $f : A \longrightarrow B$ and $g : B \longrightarrow C$ are epi, then $f; g$ is epi.*
4. *If $f : A \longrightarrow B$ and $g : B \longrightarrow C$ are mono, then $f; g$ is mono.*
5. *Given $f : A \longrightarrow B$ and $g : B \longrightarrow C$ with $f; g$ epi, then g is epi.*
6. *Given $f : A \longrightarrow B$ and $g : B \longrightarrow C$ with $f; g$ mono, then f is mono.*

In the von Neumann representation of the natural numbers as sets, the number 1 is represented as the singleton set $1 = \{\varnothing\}$. Given any set A, we can then consider the function sets $[1{\to}A]$ and $[A{\to}1]$. As the exercise below shows, the set $[A{\to}1]$ is always a singleton set. How about the set $[1{\to}A]$? Exercise 44 shows that we always have a bijection $A \cong [1{\to}A]$.

Exercise 36 *Prove the following for any set A:*

1. *The function set $[A{\to}1]$ is always a singleton set. Describe explicitly the unique function, let us denote it $!_A$, in the singleton set $[A{\to}1]$.*
2. *Prove that for all sets A, except one of them, the function $!_A$ is surjective. For which A does $!_A$ fail to be surjective? In this failure case, is $!_A$ injective? Can you give a necessary and sufficient condition on A so that $!_A$ is bijective iff your condition holds?*

Note that, since for any set A we have the set equality $\varnothing \times A = \varnothing$, then we also have the set equalities $\mathcal{P}(\varnothing \times A) = \mathcal{P}(\varnothing) = \{\varnothing\} = 1$. Furthermore, the unique relation $\varnothing : \varnothing \Longrightarrow A$ is obviously a function. As a consequence, we have the additional set equalities $\mathcal{P}(\varnothing \times A) = [\varnothing{\to}A] = \{\varnothing\} = 1$. That is, for any set A there is always a unique function from the empty set to it, namely, the function $\varnothing : \varnothing \longrightarrow A$, which is precisely the inclusion function $j^A_\varnothing : \varnothing \hookrightarrow A$, and therefore always injective. What about the function set $[A{\to}\varnothing]$? We of course have $[A{\to}\varnothing] \subseteq \mathcal{P}(A \times \varnothing) = \mathcal{P}(\varnothing) = \{\varnothing\} = 1$. But if $A \neq \varnothing$ the unique relation $\varnothing : A \Longrightarrow \varnothing$ is *not* a function. Therefore, if $A \neq \varnothing$ we have $[A{\to}\varnothing] = \varnothing$, that is, there are obviously *no* functions from a nonempty set to the empty set. What about the case $A = \varnothing$? Then we have the set equalities: $[\varnothing{\to}\varnothing] = \mathcal{P}(\varnothing \times \varnothing) = \mathcal{P}(\varnothing) = \{\varnothing\} = 1$. Furthermore, the unique function $\varnothing : \varnothing \longrightarrow \varnothing$ is precisely the identity function $id_\varnothing$.

Given any two sets A and B, we can associate to their cartesian product two functions, $p_1 : A \times B \longrightarrow A : (a, b) \mapsto a$ and $p_2 : A \times B \longrightarrow B : (a, b) \mapsto b$, which are called the *projection* functions from the cartesian product $A \times B$ into their first and second components, A and B.

Exercise 37 *The above definitions of the functions p_1 and p_2 are quite high-level, since they hide the representation of (a, b) as the set $\{\{a\}, \{a, b\}\}$. Prove that, using the concrete representation of pairs, the functions p_1 and p_2 are exactly the functions: $p_1 = \lambda x \in A{\times}B.\ \bigcup\bigcap x \in A$, and $p_2 = \lambda x \in A{\times}B.$ **if** $(\bigcup x - \bigcap x) = \varnothing$ **then** $\bigcup\bigcap x$ **else** $\bigcup(\bigcup x - \bigcap x)$ **fi** $\in B$. (Hint: use Exercise 4).*

Exercise 38 *Given any three sets A, B, and C, and given any two functions $f : C \longrightarrow A$ and $g : C \longrightarrow B$, we can define the function $(f, g) : C \longrightarrow A \times B : c \mapsto (f(c), g(c))$. Prove that:*

1. *$(f, g); p_1 = f$,*
2. *$(f, g); p_2 = g$,*
3. *(1) and (2) uniquely determine (f, g), that is, any function $h : C \longrightarrow A \times B$ such that $h; p_1 = f$ and $h; p_2 = g$ must necessarily satisfy $h = (f, g)$.*

[5] By "dually" I mean that, by "reversing the direction of all the arrows," e.g., from $A \longrightarrow B$ to $A \longleftarrow B$, in the definition of "epi" we then obtain the definition of "mono" as its "dual concept."

We can compactly express properties (1)–(3) in Exercise 38 in a precise graphical notation by means of the following *commutative diagram:*

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, dotted, "{(f,g)}"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "p_1"] \arrow[r, "p_2"'] & B
\end{tikzcd}$$

where:

- Different paths of arrows having the same beginning and ending nodes are pictorially asserted to have identical function compositions. In the above diagram the left triangle asserts equation (1), and the right triangle asserts equation (2). This is called "diagram commutativity."
- A dotted arrow pictorially denotes a *unique existential* quantification. In the above diagram the fact that the arrow for (f, g) is dotted, exactly means that (f, g) is the *unique* function that makes the two triangles commute, that is, such that (1) and (2) hold; which is statement (3).

Given any two sets A and B, we can associate to their disjoint union $A \oplus B$ two injective functions, $i_1 : A \longrightarrow A \oplus B : a \mapsto (a, 0)$ and $i_2 : B \longrightarrow A \oplus B : b \mapsto (b, 1)$, which are called the *injection* functions into the disjoint union $A \oplus B$ from their first and second components, A and B.

Exercise 39 *Given any three sets A, B, and C, and given any two functions $f : A \longrightarrow C$ and $g : B \longrightarrow C$, we can define the function $[f, g] : A \oplus B \longrightarrow C : x \mapsto$ **if** $x \in A \times \{0\}$ **then** $f(p_1(x))$ **else** $g(p_1(x))$ **fi**, where $p_1 : A \times \{0\} \longrightarrow A$, and $p_1 : B \times \{1\} \longrightarrow B$ are the projection functions.*

Prove that:

1. $i_1; [f, g] = f$,
2. $i_2; [f, g] = g$,
3. *(1) and (2) uniquely determine $[f, g]$, that is, any function $h : A \oplus B \longrightarrow C$ such that $i_1; h = f$ and $i_2; h = g$ must necessarily satisfy $h = [f, g]$.*

Properties (1)–(3) can be succinctly expressed by the commutative diagram:

$$\begin{tikzcd}
 & C & \\
A \arrow[ur, "f"] \arrow[r, "i_1"'] & A \oplus B \arrow[u, dotted, "{[f,g]}"] & B \arrow[ul, "g"'] \arrow[l, "i_2"]
\end{tikzcd}$$

Note the striking similarity between Exercises 38 and 39, since, except for "reversing the direction of all the arrows," the product $A \times B$ and the disjoint union $A \oplus B$ have the exact same properties with respect to functions from a set C to their components A and B (resp., to a set C from their components A and B). As already mentioned for epis and monos in Footnote 5 to Execise 35, this striking coincidence, obtained by "reversing the direction of the arrows" is called *duality*. Therefore, the cartesian product $A \times B$ and the disjoint union $A \oplus B$ are *dual* constructions. For this reason, the disjoint union is sometimes called the *coproduct* of A and B.

In fact, this kind of duality is at work not just between epis and monos and between products and disjoint unions. In a similar way, the empty set $\varnothing$ and the set 1 are dual of each other, since for any set A, the function sets $[\varnothing \to A]$ and $[A \to 1]$ are both singleton sets. That is, for any set A there is a *unique* function $\varnothing : \varnothing \longrightarrow A$ from $\varnothing$ to A, and, dually, there is also a *unique* function $!_A : A \longrightarrow 1$ from A to 1. That is, "reversing the direction of the arrows" $\varnothing$ and 1 behave just the same. For this reason, $\varnothing$ is sometimes called the *initial* set, and its dual, 1, the *final* set.

6.5 Abstract Products and Disjoint Unions

Exercises 39 and 38 are much more than just two average exercises: they give us the key for arriving at the *abstract* notions of "disjoint union" and "cartesian product," thus freeing them of any "representation bias."

Let me begin with cartesian products and explain how we can use the properties stated in Exercise 38 to arrive at the right abstract notions of "cartesian product" and "ordered pair." As we discussed in §4.1, the set-theoretic representation of an ordered pair as $(x, y) = \{\{x\}, \{x, y\}\}$ is just *one choice* of representation of ordered pairs among many. Since cartesian products are defined as sets of ordered pairs, our definition of cartesian product is based on the above representation of ordered pairs, and is also *one choice* of representation of cartesian products among many. But how could we *characterize* all the possible correct representations of the ordered pair and cartesian product notions? And how could we precisely express the corresponding *abstract concepts* for such notions within our set theory language? We can do all this very easily by turning Exercise 38 on its head and using it as the *abstract definition* of a cartesian product. That is, we can ignore for the moment our earlier representations for ordered pairs and cartesian products and adopt the following definition:

Definition 11 *(Abstract Product) Given two sets A and B, a* cartesian product *for A and B is a set, denoted $A \times B$, together with two functions $p_1 : A \times B \longrightarrow A$ and $p_2 : A \times B \longrightarrow B$, satisfying the following property: for any other set C, and any two functions $f : C \longrightarrow A$ and $g : C \longrightarrow B$, there exists a function, which we denote $(f, g) : C \longrightarrow A \times B$, such that:*

1. $(f, g); p_1 = f$,
2. $(f, g); p_2 = g$,
3. *(1) and (2) uniquely determine (f, g), that is, any other function $h : C \longrightarrow A \times B$ such that $h; p_1 = f$ and $h; p_2 = g$ must necessarily satisfy $h = (f, g)$.*

In view of this abstract definition, Exercise 38 now becomes just *checking* that our concrete choice of representation for ordered pairs and cartesian products is a *correct representation* of the abstract concept.

But, since Definition 11 does not tell us *anything* about the internal representation of the elements in the set $A \times B$, it describes such set as what, both in object-oriented programming and in the theory of algebraic data types, is called an *abstract data type*: it makes the internal representation of ordered pairs *hidden*, giving the "implementer" enormous freedom to internally represent ordered pairs as he/she chooses. Any representation will be correct if and only if it satisfies the requirements specified in the abstract definition.

For example, instead of the standard representation of ordered pairs that we have used so far, we could have chosen the alternative representation discussed in Exercise 3. With the appropriate definition of p_1 and p_2 for this alternative representation (as an exercise, give concrete definions of p_1 and p_2 for this new representation in a way entirely similar to Exercise 37), this does of course also satisfy the abstract requirements in Definition 11. A more striking example of a very different choice of representation for ordered pairs and the corresponding cartesian product is provided by the *Gödel numbering* technique used —not only in logic, but also in cryptography— for encoding pairs and, more generally, terms in a language as numbers. Suppose that we want to represent the cartesian product $\mathbb{N} \times \mathbb{N}$ this way. We can do so by defining for any two numbers $n, m \in \mathbb{N}$ their ordered pair as the number: $(n, m) = 2^{n+1} \cdot 3^{m+1}$. Then we can define $\mathbb{N} \times \mathbb{N} = \{2^{n+1} \cdot 3^{m+1} \in \mathbb{N} \mid n, m \in \mathbb{N}\}$, and give to p_1 and p_2 the obvious definitions: $p_1 : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N} : 2^{n+1} \cdot 3^{m+1} \mapsto n$, $p_2 : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N} : 2^{n+1} \cdot 3^{m+1} \mapsto m$, where the correctness of this representation of pairs and the well-definedness of the functions p_1 and p_2 are trivial consequences of the Fundamental Theorem of Arithmetic, that ensures that any natural number greater than 1 has a *unique factorization* as a product of powers of its prime factors. Given functions $f, g : C \longrightarrow \mathbb{N}$, the unique function $(f, g) : C \longrightarrow \mathbb{N} \times \mathbb{N}$ such that $(f, g); p_1 = f$ and $(f, g); p_2 = g$ is then given by the assignment: $c \mapsto 2^{f(c)+1} \cdot 3^{g(c)+1}$.

Note that the abstract notion of "ordered pair" and its extensionality property (Lemma 1) now appear as trivial consequences of the abstract notion of "cartesian product." Given any set A, we can exploit the bijection $A \cong [1 \to A] : a \mapsto \widehat{a}$, explored in detail in Exercise 44, to faithfully represent any element $a \in A$ as a function $\widehat{a} : 1 \longrightarrow A : 0 \mapsto a$. Then, specializing the set C to 1 in Definition 11, what we get is that for any two functions $\widehat{a} : 1 \longrightarrow A$, $\widehat{b} : 1 \longrightarrow B$, there exists a unique function $(\widehat{a}, \widehat{b}) : 1 \longrightarrow A \times B$ such that $(\widehat{a}, \widehat{b}); p_1 = \widehat{a}$ and $(\widehat{a}, \widehat{b}); p_2 = \widehat{b}$. Defining an *ordered pair* as an element $(a, b) = (\widehat{a}, \widehat{b})(0) \in A \times B$, this trivially implies that the extensionality Lemma 1 holds for *any* representation of pairs provided by *any* abstract product. This is of course an *enormously more general* extensionality result than Lemma 1. Furthermore, it has a *much simpler proof* (just given above!) than that of Lemma 1. This is just one instance

of what I like to call the "generalize and conquer" principle, where many times the more general a result is, the simpler its proof becomes!

Having identified the duality between products and disjoint unions provides a great economy of thought, because we then get two abstract concepts for the price of one. That is, we get the concept of "abstract disjoint union," just by reversing the arrows in Definition 11:

Definition 12 *(Abstract Disjoint Union) Given two sets A and B, a* disjoint union *for A and B is a set, denoted $A \oplus B$, together with two functions $i_1 : A \longrightarrow A \oplus B$ and $i_2 : B \longrightarrow A \oplus B$, satisfying the following property: for any other set C, and any two functions $f : A \longrightarrow C$ and $g : A \longrightarrow C$, there exists a function, which we denote $[f, g] : A \oplus B \longrightarrow C$, such that:*

1. $i_1; [f, g] = f$,
2. $i_2; [f, g] = g$,
3. *(1) and (2) uniquely determine $[f, g]$, that is, any function $h : A \oplus B \longrightarrow C$ such that $i_1; h = f$ and $i_2; h = g$ must necessarily satisfy $h = [f, g]$.*

As before, Exercise 39 now becomes checking that our concrete representation for $A \oplus B$ as the set $(A \times \{0\}) \cup (B \times 1)$, together with our choice of inclusion functions i_1 and i_2 for $(A \times \{0\}) \cup (B \times 1)$ is a *correct representation* of the abstract concept of disjoint union.

But once we have the abstract concept, there is again great freedom from any "representation bias;" because we do not need to *care* about how the elements inside $A \oplus B$ are internally represented: we just treat $A \oplus B$ as an *abstract* data type. There are, indeed, many possibilities. For example, whenever $A \cap B = \varnothing$ we can choose $A \oplus B = A \cup B$, with i_1 and i_2 the inclusion functions $j_A^{A \cup B} : A \hookrightarrow A \cup B$ and $j_B^{A \cup B} : B \hookrightarrow A \cup B$, which trivially satisfies properties (1)–(3) in Definition 12 by taking $[f, g] = f \cup g$. For another example, we can choose a disjoin union representation $\mathbb{N} \oplus \mathbb{N} = \mathbb{N}$, with i_1 and i_2, the functions: $2 \cdot \_ : \mathbb{N} \longrightarrow \mathbb{N}$ and $1 + (2 \cdot \_) : \mathbb{N} \longrightarrow \mathbb{N}$. It is trivial to check that this gives us a correct representation of the abstract notion of disjoint union $\mathbb{N} \oplus \mathbb{N}$, with $[f, g](n) =$ **if** $even(n)$ **then** $f(n/2)$ **else** $g((n-1)/2)$ **fi**.

Anybody in his right mind would expect that all these different representations of an abstract product $A \times B$, (resp., an abstract disjoint union $A \oplus B$) are "essentially the same." This is actually the case, not only in the weak sense that there are *bijections* between these different representations, but, as shown in Exercise 40 below, in the much stronger sense that those bijections preserve what in computer science are called the "interfaces" of the corresponding abstract data types. For products the "interface" is given by the projection functions p_1 and p_2, whereas for disjoint union the "interface" is given by the injection functions i_1 and i_2. Such bijections preserving the interfaces are called *isomorphisms*.

Exercise 40 *(All products (resp., disjoint unions) of A and B are isomorphic). Prove that:*

1. *Given any two sets A, B, and given two different representations $A \times B$ and $A \times' B$ of the abstract product of A and B, with projections $p_1 : A \times B \longrightarrow A$ and $p_2 : A \times B \longrightarrow B$, resp., $p'_1 : A \times' B \longrightarrow A$ and $p'_2 : A \times' B \longrightarrow B$, there is a unique bijection $h : A \times B \xrightarrow{\cong} A \times' B$ such that: (i) $h; p'_1 = p_1$, and $h; p'_2 = p_2$; and (ii) $h^{-1}; p_1 = p'_1$, and $h^{-1}; p_2 = p'_2$.*
2. *State and prove the dual statement for disjoint unions. Can you "reuse" your proof of (1) to obtain "automatically" a proof of (2), just by reversing the arrows?*

Exercise 41 *($\oplus$ and $\times$ are Functorial Constructions). For concreteness we assume the standard representations for $A \oplus B$ and $A \times B$. These are set-theoretic constructions that act naturally not only on sets A and B, but also on functions between sets. For example, given functions $f : A \longrightarrow A'$ and $g : B \longrightarrow B'$, we can define the function $f \oplus g : A \oplus B \longrightarrow A' \oplus B'$ by the defining equation $f \oplus g = [f; i_1, g; i_2]$. Applying the definition of $[f; i_1, g; i_2]$, we easily see that $f \oplus g$ behaves like f on the disjoint copy of A and like g on the disjoint copy of B. That is, $(f \oplus g)(a, 0) = (f(a), 0)$, and $(f \oplus g)(b, 1) = (g(b), 1)$. Dually, by replacing injections by projections and inverting the direction of the functions, we can define the function $f \times g : A \times B \longrightarrow A' \times B'$ by the defining equation $f \times g = (p_1; f, p_2; g)$. Applying the definition of $(p_1; f, p_2; g)$, we easily see that $f \times g$ behaves like f on the first coordinate, and like g on the second, that is, $(f \times g)(a, b) = (f(a), g(b))$.*

Any set-theoretic construction is called functorial *if it satisfies three properties. First, it* acts not only on sets, but also on functions relating such sets*, so that the constructed function then relates the sets so constructed. We have already checked this first property for $\oplus$ and $\times$ by our definitions for $f \oplus g$ and $f \times g$. Second, it* preserves function

composition. For $\oplus$ and $\times$ this means that if we also have functions $f' : A' \longrightarrow A''$ and $g' : B' \longrightarrow B''$, then we have equalities

$$(f \oplus g); (f' \oplus g') = (f; f') \oplus (g; g') \qquad \textit{resp.,} \qquad (f \times g); (f' \times g') = (f; f') \times (g; g').$$

Third, it preserves identity functions. *For $\oplus$ and $\otimes$ this means that for any two sets A and B we have*

$$id_A \oplus id_B = id_{A \oplus B} \qquad \textit{resp.,} \qquad id_A \times id_B = id_{A \times B}.$$

Prove that the above four equalities hold, and therefore that $\oplus$ and $\times$ are indeed functorial constructions.

Exercise 42 *(***if-then-else-fi***) Let $f, g : A \longrightarrow B$ be any two functions from A to B, and let $\varphi(x)$ be a formula in the language of set theory having a single free variable x. Prove that there is a unique function, denoted* **if** φ **then** f **else** g **fi**, *such that for each $a \in A$,* (**if** φ **then** f **else** g **fi**)$(a) = f(a)$ *if $\varphi(a)$ holds, and* (**if** φ **then** f **else** g **fi**)$(a) = g(a)$ *if $\varphi(a)$ doesn't hold. (Hint: decompose A as an abstract disjoint union.)*

6.6 Relating Function Sets

We can view relation and function composition as *functions*. For example, composing relations from A to B with relations from B to C, is the function

$$\_;\_ : \mathcal{P}(A \times B) \times \mathcal{P}(B \times C) \longrightarrow \mathcal{P}(A \times C) : (F, G) \mapsto F; G$$

Similarly, composing functions from A to B with functions from B to C, is the function

$$\_;\_ : [A{\to}B] \times [B{\to}C] \longrightarrow [A{\to}C] : (f, g) \mapsto f; g$$

Also, given sets A, B and B', and a function $g : B \longrightarrow B'$, we can define the function

$$[A{\to}g] : [A{\to}B] \longrightarrow [A{\to}B'] : h \mapsto h; g$$

Likewise, given sets A, A' and B, and a function $f : A' \longrightarrow A$, we can define the function

$$[f{\to}B] : [A{\to}B] \longrightarrow [A'{\to}B] : h \mapsto f; h$$

More generally, given sets A, A', B and B', and functions $f : A' \longrightarrow A$ and $g : B \longrightarrow B'$, we can define the function

$$[f{\to}g] : [A{\to}B] \longrightarrow [A'{\to}B'] : h \mapsto f; h; g$$

so that we then get $[A{\to}g] = [id_A{\to}g]$, and $[f{\to}B] = [f{\to}id_B]$ as special cases.

Exercise 43 *($[\_{\to}\_]$ is a Functorial Construction). The above definition of $[f{\to}g]$ strongly suggests that $[\_{\to}\_]$ acts on both sets and functions and is a functorial construction. However, in the case of the $[\_{\to}\_]$ construction, its action on functions comes with a* twist, *since $[f{\to}g]$* reverses the direction of the function in its first argument: *we give it $f : A' \longrightarrow A$, but we get back $[f{\to}B] : [A{\to}B] \longrightarrow [A'{\to}B]$. This twist is called being "contravariant" on the first argument. Prove that $[\_{\to}\_]$ satisfies the other two functoriality requirements: it preserves both function composition and identity functions. Because of the contravariance on the first argument, what now has to be proved is that given functions*

$$A \xleftarrow{f} A' \xleftarrow{f'} A'' \qquad\qquad B \xrightarrow{g} B' \xrightarrow{g'} B''$$

we have $[f{\to}g]; [f'{\to}g'] = [(f'; f){\to}(g; g')]$. The requirement for identity preservation is as expected: given any two sets A and B one has to prove the equality $[id_A{\to}id_B] = id_{[A{\to}B]}$.

For any function set $[B{\to}C]$, *function evaluation* is itself a function

$$\_(\_)_{[B{\to}C]} : [B{\to}C] \times B \longrightarrow C : (f, b) \mapsto f(b).$$

Also, for any cartesian product $A \times B$, we have a function

$$column_{A \times B} : A \longrightarrow [B{\to}(A \times B)] : a \mapsto \lambda y \in B.\ (a, y) \in A \times B.$$

Note that the name $column_{A \times B}$ is well-chosen, since if we visualize the cartesian product $A \times B$ as a two-dimensional table, where each $a \in A$ gives rise to the "column" $\{(a, y) \in A \times B \mid y \in B\}$, and each $b \in B$ gives rise to the "row" $\{(x, b) \in A \times B \mid x \in A\}$, then $column_{A \times B}(a)$ maps each $y \in B$ to its corresponding position (a, y) in the column $\{(a, y) \in A \times B \mid y \in B\}$.

Exercise 44 *Prove the following for any set A:*

1. *The first projection map* $p_A : A \times 1 \longrightarrow A : (a, \emptyset) \mapsto a$ *is bijective.*
2. *The evaluation function* $\_(\_)_{[1 \to A]} : [1 \to A] \times 1 \longrightarrow A$ *is bijective.*
3. *Combine (1) and (2) to prove that the function* $[1 \to A] \longrightarrow A : f \mapsto f(\emptyset)$ *is bijective. That is, for all practical purposes we can think of an element* $a \in A$ *as a function* $\widehat{a} \in [1 \to A]$*, namely the unique function* $\widehat{a}$ *such that* $\widehat{a}(\emptyset) = a$*, and, conversely, any function* $f \in [1 \to A]$ *is precisely of the form* $f = \widehat{a}$ *for a unique* $a \in A$*.*

Exercise 45 *Prove that for any set A there is a bijection* $A^2 \cong [2 \to A]$*. That is, except for a slight change of representation, the cartesian product* $A \times A$ *and the function set* $[2 \to A]$ *are "essentially the same set."*

Generalize this for any $n \geq 1$*, adopting the notational convention that for* $n = 1$*, A coincides with the "1-fold cartesian product" of A. That is, show that for any* $n \geq 1$ *there is a bijection* $A^n \cong [n \to A]$*, between the n-fold cartesian product of A (as defined in §4 for* $n \geq 2$*, and here also for* $n = 1$*), and the function set* $[n \to A]$*. Note that for* $n = 1$ *this yields the bijection between A and* $[1 \to A]$ *already discussed in Exercise 44-(3).*

Regarding the above exercise, note that if A is a finite set with m elements, then, recalling Exercise 16, the n-fold product of A has $m \cdot \overset{n}{\ldots} \cdot m$ elements, and the function set $[n \to A]$ has (how many? See Exercise 28). Therefore, the above bijection $A^n \cong [n \to A]$ tells us two things. First, that the same way that the cartesian product construction generalizes number multiplication, the function set construction generalizes number *exponentiation*. Second, that the bijection $A^n \cong [n \to A]$ generalizes the arithmetic identity $m \cdot \overset{n}{\ldots} \cdot m = m^n$.

Exercise 46 *(Currying and un-Currying of Functions). Prove that for any three sets A, B and C, the function*

$$curry : [A \times B \to C] \longrightarrow [A \to [B \to C]] : f \mapsto column_{A \times B}; [B \to f]$$

is bijective, since it has as its inverse the function

$$curry^{-1} : [A \to [B \to C]] \longrightarrow [A \times B \to C] : h \mapsto (h, p_2); \_(\_)_{[B \to C]}$$

where $(h, p_2) : A \times B \longrightarrow [B \to C] \times B$ *is the unique function associated to h and* p_2 *in Exercise 38.*

Currying (after Haskell Curry) allows us to transform a function $f : A \times B \longrightarrow C$ of two arguments into a higher-order-valued function $curry(f) : A \longrightarrow [B \to C]$ of its first argument. Given an element $a \in A$, then $curry(f)(a) = \lambda x \in B.\ f(a, x) \in C$. Therefore, for any $(x, y) \in A \times B$ we have the equality, $curry(f)(x)(y) = f(x, y)$. Un-currying is the inverse transformation, that brings down a higher-order-valued function of this type into a function of two arguments. Therefore, for any $h \in [A \to [B \to C]]$ and any $(x, y) \in A \times B$ we have the equality, $curry^{-1}(h)(x, y) = h(x)(y)$. In functional programming, currying can be used in combination with the technique called "partial evaluation" to speed up function evaluation. The idea is that if $f : A \times B \longrightarrow C$ is a function of two arguments, but we know that its first argument in a certain situation will always be a fixed value a, we may be able to use symbolic evaluation to derive a specialized algorithm for the one-argument function $curry(f)(a)$ that is more efficient than the general algorithm available for f.

We finish this section with an exercise exploring in depth the relationships between the set of relations $\mathcal{P}(A \times B)$ and the set of functions $[A \to \mathcal{P}(B)]$.

Exercise 47 *(Relations as Functions). Given a relation* $F : A \Longrightarrow B$*, we can associate to it the function* $\widetilde{F} : A \longrightarrow \mathcal{P}(B) : a \mapsto F[\{a\}]$*. Of course, F and* $\widetilde{F}$ contain the same information*, since* any *function* $f : A \longrightarrow \mathcal{P}(B)$ *is always of the form* $f = \widetilde{F}$ *for a* unique *relation F, namely, for* $F = \{(a, b) \in A \times B \mid b \in f(a)\}$*. Prove that the mapping* $\widetilde{(\_)} : \mathcal{P}(A \times B) \longrightarrow [A \to \mathcal{P}(B)] : F \mapsto \widetilde{F}$ *is in fact a* bijection.

Note that if we have relations $F : A \Longrightarrow B$ *and* $G : B \Longrightarrow C$*, we can compose them to obtain* $F; G : A \Longrightarrow C$*, but* $\widetilde{F}$ *and* $\widetilde{G}$ *cannot be composed anymore in the standard way, since their domains and codomains* do not match*. However, we can give a* different definition of composition *so that they* do *compose, in a way that mimics perfectly the composition of their corresponding relations F and G. Specifically, we can define the following new composition operation:*

$$\widetilde{F} \star \widetilde{G} = \widetilde{F}; (G[\_])$$

where composition in the right side of the equality is the usual function composition, since now the codomain of $\widetilde{F} : A \longrightarrow \mathcal{P}(B)$ *and the domain of* $G[\_] : \mathcal{P}(B) \longrightarrow \mathcal{P}(C)$ do *match.*

Note, finally, that for id_A *the identity function the corresponding function* $\widetilde{id_A} : A \longrightarrow \mathcal{P}(A)$ *is just the function* $\{\_\}_A : A \longrightarrow \mathcal{P}(A) : a \mapsto \{a\}$*, mapping each* $a \in A$ *to the corresponding singleton set* $\{a\}$*.*

Prove the following:

1. *This composition* mimics perfectly relation composition*, that is, we have the equality* $\widetilde{F; G} = \widetilde{F} \star \widetilde{G}$*.*
2. *It is* associative*, that is, given* $F : A \Longrightarrow B$*,* $G : B \Longrightarrow C$*, and* $H : C \Longrightarrow D$*, we have:* $\widetilde{F} \star (\widetilde{G} \star \widetilde{H}) = (\widetilde{F} \star \widetilde{G}) \star \widetilde{H}$*.*
3. *The maps* $\widetilde{id_A} = \{\_\}_A$ *act as* identities*, that is, given* $\widetilde{F} : A \Longrightarrow \mathcal{P}(B)$ *we have the equalities* $\{\_\}_A \star \widetilde{F} = \widetilde{F} = \widetilde{F} \star \{\_\}_B$*.*

Chapter 7

Simple and Primitive Recursion, and the Peano Axioms

The natural numbers can be used to define recursive functions by simple recursion and by primitive recursion. They satisfy some simple axioms, due to Giuseppe Peano and reformulated by F. William Lawvere, which are intimately related to simple recursion.

7.1 Simple Recursion

We routinely define all kinds of functions inductively. For example, consider the function $double : \mathbb{N} \longrightarrow \mathbb{N} : n \mapsto 2 \cdot n$, which doubles each natural number. We can define it inductively in terms of the successor function by means of the inductive definition:

- $double(0) = 0$
- $double(s(n)) = s(s(double(n))$.

Such a definition is just a special case of a general way of defining functions inductively by *simple recursion*. The general idea is as follows. We are given a set A, an element $a \in A$, and a function $f : A \longrightarrow A$. Then these data always define a function $rec(f, a) : \mathbb{N} \longrightarrow A$ that satisfies:

- $rec(f, a)(0) = a$
- $rec(f, a)(s(n)) = f(rec(f, a)(n))$.

Why is this, admittedly quite intuitive, way of inductively defining functions *sound*? That is, how do we *know* that such a method uniquely defines a function? The answer is that this is a consequence of the Peano Induction (Theorem 2), as the following theorem shows.

Theorem 5 *(Simple Recursion). For any set A, element $a \in A$, and function $f : A \longrightarrow A$, there exists a unique function $rec(f, a) : \mathbb{N} \longrightarrow A$ such that:*

- $rec(f, a)(0) = a$
- $rec(f, a)(s(n)) = f(rec(f, a)(n))$.

Proof. We first prove that such a function exists, and then that it is unique. Let $\mathcal{R}(f, a)$ be the set of all relations $R \subseteq \mathbb{N} \times A$ such that:

1. $(0, a) \in R$
2. $(\forall n \in \mathbb{N})\ ((n, a') \in R \ \Rightarrow (s(n), f(a')) \in R)$.

Since $(\mathbb{N} \times A) \in \mathcal{R}(f,a)$, $\mathcal{R}(f,a)$ is nonempty. Let $rec(f,a) = \bigcap \mathcal{R}(f,a)$. It is trivial to check that, by construction, $rec(f,a)$ satisfies (1)–(2). We will be done with the existence part of the proof if we show that $rec(f,a)$ is a function. Let $X \subseteq \mathbb{N}$ be the set of natural numbers n such that the set $rec(f,a)(n)$ is a singleton set. If we show $X = \mathbb{N}$ we will be done with the existence part. But by Peano Induction, to show $X = \mathbb{N}$ it is enough to show that X is a successor set. We reason by contradiction. Suppose that $0 \notin X$. Then we must have $(0,a),(0,a') \in rec(f,a)$, with $a \neq a'$. But then it is trivial to show that $(rec(f,a) - \{(0,a')\}) \in \mathcal{R}(f,a)$, against the minimality of $rec(f,a)$. Suppose, instead, that there is an $n \in X$ such that $s(n) \notin X$, and let (n,a') be the unique pair in $rec(f,a)$ with n its first component. Then we must have $(s(n), f(a')), (s(n), a'') \in rec(f,a)$, with $f(a') \neq a''$. But then it is trivial to show that $(rec(f,a) - \{(s(n), a'')\}) \in \mathcal{R}(f,a)$, against the minimality of $rec(f,a)$.

To prove uniqueness, suppose that there is another function, say, $rec'(f,a)$, satisfying the conditions stated in the theorem. Then we must have $rec'(f,a) \in \mathcal{R}(f,a)$, which forces the set-theoretic containment $rec(f,a) \subseteq rec'(f,a)$. But this implies $rec(f,a) = rec'(f,a)$, since, in general, given any two sets X and Y, and any two functions $f, f' \in [X \to Y]$, whenever $f \subseteq f'$ we must have $f = f'$. □

The Simple Recursion Theorem can be precisely understood as the statement that the set $\mathbb{N}$ of von Neumann natural numbers satisfies the following more general axiom due to F. William Lawvere [31]. Recall from Exercise 44 that an element $a \in A$ can be alternatively represented as a function $\widehat{a} : 1 \longrightarrow A : \emptyset \mapsto a$. Using this representation of set elements as functions, the Simple Recursion Theorem is then the statement that the von Neumann naturals satisfy the following axiom, called the *Peano-Lawvere* axiom.

> *(Peano-Lawvere) There exists a set $\mathbb{N}$ and functions $\widehat{0} : 1 \longrightarrow \mathbb{N}$ and $s : \mathbb{N} \longrightarrow \mathbb{N}$, such that given any set A, and functions $\widehat{a} : 1 \longrightarrow A$ and $f : A \longrightarrow A$, there is a unique function $rec(f,a) : \mathbb{N} \longrightarrow A$ making the triangle and the square commute in the diagram:*

$$\begin{tikzcd}
 & \mathbb{N} \arrow[r, "s"] \arrow[dd, dotted, "rec(f,a)"] & \mathbb{N} \arrow[dd, dotted, "rec(f,a)"] \\
1 \arrow[ur, "\widehat{o}"] \arrow[dr, "\widehat{a}"'] & & \\
 & A \arrow[r, "f"'] & A
\end{tikzcd}$$

> *that is, such that $\widehat{a}; rec(f,a) = \widehat{0}$, and $rec(f,a); f = s; rec(f,a)$.*

Obviously, Theorem 5 can be equivalently reformulated as the statement that the set $\mathbb{N}$ of the von Neumann natural numbers, guaranteed to exist as a set by the (*Inf*) axiom, together with the functions $\widehat{0} : 1 \longrightarrow \mathbb{N}$, and $s : \mathbb{N} \longrightarrow \mathbb{N} : n \mapsto n \cup \{n\}$ satisfies the *Peano-Lawvere* axiom. But note that the *Peano-Lawvere* axiom is completely *abstract*, and does not depend on any particular representation of the natural numbers, such as the von Neumann representation. For example, the set $\mathbb{N}$ claimed to exist by *Peano-Lawvere* could be the binary or decimal representation of the naturals, with s the function $s : n \mapsto n + 1$; or it could be the Zermelo encoding $0 = \emptyset$, $1 = \{\emptyset\}$, $2 = \{\{\emptyset\}\}$, ..., etc., with $s : x \mapsto \{x\}$. This abstract formulation is actually better than having a specific representation of the natural numbers, such as the von Neumann representation, which is implicitly built into the (*Inf*) axiom because of its formulation in terms of successor sets. The reason why a more abstract axiomatization is better than a more concrete one is that then the properties that follow from the given axioms can be directly applied to *any* set satisfying the axioms, and not just to a particular set encoding a particular representation.

Note the striking similarity of the Peano-Lawvere *abstract* definition of the natural numbers as an abstract data type, in which we do not have to *care* about the internal representation chosen for numbers, and the concepts of abstract product and abstract disjoint union defined in §6.5, where, similarly, we viewed a product or a disjoint union as an abstract data type and became liberated from any representation bias. In Exercise 40 we saw that all representations of an abstract product $A \times B$ (resp., an abstract disjoint union $A \oplus B$) were" isomorphic," in the precise sense that there was a unique bijection between any two such representations "preserving the interfaces," i.e., preserving the projections (resp., injections). For the abstract data type of the natural numbers, the "interface" is of course given by the functions $\widehat{0} : 1 \longrightarrow \mathbb{N}$ and

$s : \mathbb{N} \longrightarrow \mathbb{N}$. Again, as anybody in his right mind would expect, all such representations are *isomorphic*, in the precise sense that there is a unique bijection preserving the interfaces.

Exercise 48 *(All natural number representations are isomorphic). Let $\mathbb{N}$ and $\mathbb{N}'$ be two representations of the natural numbers satisfying the properties asserted in the* Peano-Lawvere *axiom with associated functions $\widehat{0} : 1 \longrightarrow \mathbb{N}$ and $s : \mathbb{N} \longrightarrow \mathbb{N}$, and $\widehat{0'} : 1 \longrightarrow \mathbb{N}'$ and $s' : \mathbb{N}' \longrightarrow \mathbb{N}'$. Prove that there is a unique bijection $h : \mathbb{N} \xrightarrow{\cong} \mathbb{N}'$ such that: (i) $\widehat{0}; h = \widehat{0'}$, and $h; s' = s; h$; and (ii) $\widehat{0'}; h^{-1} = \widehat{0}$, and $h^{-1}; s = s'; h^{-1}$.*

Exercise 49 *(Peano-Lawvere implies Induction). Prove that any set $\mathbb{N}$ with associated functions $\widehat{0} : 1 \longrightarrow \mathbb{N}$ and $s : \mathbb{N} \longrightarrow \mathbb{N}$ satisfying the Peano-Lawvere axiom also satisfies the following induction property: $(\forall T \subseteq \mathbb{N})\ ((0 \in T \ \wedge\ ((\forall x \in T)\ s(x) \in T)) \ \Rightarrow T = \mathbb{N})$.*

7.2 Primitive Recursion

Consider the following recursive definition of the addition function:

1. 0 + m = m
2. s(n) + m = s(n + m).

The syntactic format of this definition does not fit the format of definitions by simple recursion. It is instead an instance of a general method for defining functions called *primitive recursion*. The general scheme is as follows. We are given two sets A and B, and two functions $g : B \longrightarrow A$ and $f : A \longrightarrow A$. Then we define a function $prec(f, g) : \mathbb{N} \times B \longrightarrow A$ as the unique function obtained from f and g that satisfies the equations:

1. $prec(f, g)(0, b) = g(b)$
2. $prec(f, g)(s(n), b) = f(prec(f, g)(n, b))$

For the case of the addition function we have $A = B = \mathbb{N}$, $g = id_{\mathbb{N}}$, and $f = s$, so $+ = prec(s, id_{\mathbb{N}})$.

Note that, in essence, primitive recursion contains simple recursion as a special case, namely, the case when $B = 1$. I say, "in essence" because given $\widehat{a} : 1 \longrightarrow A$ and $f : A \longrightarrow A$, the function $prec(f, \widehat{a})$ is not exactly the function $rec(f, a) : \mathbb{N} \longrightarrow A$, since it has a slightly different typing, namely, $prec(f, \widehat{a}) : \mathbb{N} \times 1 \longrightarrow A$. But we can use the bijection $k : \mathbb{N} \xrightarrow{\cong} \mathbb{N} \times 1 : n \mapsto (n, \emptyset)$ to obtain simple recursion as a special case of primitive recursion, namely, $rec(f, a) = k; prec(f, \widehat{a})$.

Two obvious questions to ask are: (i) is primitive recursion a *sound* method for defining functions, that is, does it always define a function, and a unique function? and (ii) is primitive recursion *essentially more general* than simple recursion, that is, are there functions that we can define from some basic set of functions by primitive recursion but we cannot define by simple recursion? These two questions are answered (with respective "yes" and "no" answers) in one blow by the proof of following theorem:

Theorem 6 *(Primitive Recursion). For any sets A and B, and functions $g : B \longrightarrow A$ and $f : A \longrightarrow A$, there exists a unique function $prec(f, g) : \mathbb{N} \times B \longrightarrow A$ such that:*

1. $prec(f, g)(0, b) = g(b)$
2. $prec(f, g)(s(n), b) = f(prec(f, g)(n, b))$

Proof. We can reduce primitive recursion to simple recursion as follows. Since $g \in [B \to A]$, and f induces a function $[B \to f] : [B \to A] \longrightarrow [B \to A]$, simple recursion ensures the existence of a unique function $rec([B \to f], g) : \mathbb{N} \longrightarrow [B \to A]$ such that: (i) $rec([B \to f], g)(0) = g$, and (ii) $rec([B \to f], g)(s(n)) = rec([B \to f], g)(n); f$. We can then define $prec(f, g) : \mathbb{N} \times B \longrightarrow A$ by "un-currying" $rec([B \to f], g)$ (see Exercise 46). That is, we define $prec(f, g) = curry^{-1}(rec([B \to f], g))$. It is then an easy exercise in currying and uncurrying to check that $prec(f, g)$ satisfies (1)–(2) iff $rec([B \to f], g)$ satisfies (i)–(ii). □

The claim that simple recursion is as general a method as primitive recursion for defining some new functions out of some preexisting ones is clearly vindicated by the proof of the above theorem, *provided we allow ourselves the freedom of currying and uncurrying functions*. For example, we can define the addition function on the natural numbers by simple recursion as the function $+ = curry^{-1}(rec([\mathbb{N} \to s], id_{\mathbb{N}}))$.

Exercise 50 *Define natural number multiplication by primitive recursion using the primitive recursive definition of addition as a basis. Give then a recursive definition of natural number exponentiation based on the previously defined multiplication function.*

Primitive recursion is obviously an *effective* method to define *computable functions* (intuitively, functions that we can program in a digital computer) out of simpler ones. But not all total computable functions on the natural numbers can be defined by primitive recursion (plus function composition and *tupling*, that is, functions induced into cartesian products as explained in Exercise 38) from 0, the successor function, the identity function $id_{\mathbb{N}} : \mathbb{N} \longrightarrow \mathbb{N}$, and the projection functions $p_i : \mathbb{N}^n \longrightarrow \mathbb{N} : (x_1, \ldots, x_n) \mapsto x_i$, $n \geq 2$, $1 \leq i \leq n$. A paradigmatic example of a total computable function that cannot be defined this way is Ackermann's function:

$$\begin{aligned} A(0, n) &= s(n) \\ A(s(m), 0) &= A(m, 1) \\ A(s(m), s(n)) &= A(m, A(s(m), n)). \end{aligned}$$

However, the claim that Ackermann's function cannot be defined by primitive recursion (proved in detail in, e.g., [27], §13) has to be carefully qualified, since it essentially depends on *explicitly forbidding the currying and uncurrying of functions*, so that the only functions we ever define are always of the form $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ for some $n \in \mathbb{N}$. Instead, if we allow ourselves the freedom to curry and uncurry functions— that is, besides composition, tupling, identities, and projections, we also allow functional application functions of the form $\_(\_)$, as well as functions of the form *column*, and $[f{\to}g]$, where f and g have already been defined— it then becomes relatively easy to define Ackermann's function by primite recursion from 0 and the successor function as follows.

Lemma 5 *If currying and uncurrying of functions (plus composition, tupling, identities, projections, $\_(\_)$, column, and $[f{\to}g]$ for previously defined f, g), is allowed in the sense explained above, then Ackermann's function can be defined by primitive recursion from 0 and the successor function.*

Proof. First of all, we can obtain $A : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ by uncurrying its curried version $\widehat{A} = curry(A) : \mathbb{N} \longrightarrow [\mathbb{N}{\to}\mathbb{N}]$. Second, I claim that $\widehat{A}$ is just the simple recursive (and a fortiori primitive recursive) function $\widehat{A} = rec(Iter, s)$, where *Iter* is the iterator function $Iter : [\mathbb{N}{\to}\mathbb{N}] \longrightarrow [\mathbb{N}{\to}\mathbb{N}] : f \mapsto \lambda n.\, f^{s(n)}(1)$. This claim is easy to prove by observing that: (i) by definition, $\widehat{A}(n)(m) = A(n, m)$; and (ii) *Iter* satisfies the equation $Iter(f)(s(n)) = f(Iter(f)(n))$. We can then prove the equality $\widehat{A} = rec(Iter, s)$ by applying the definitions of $\widehat{A}(0)$ and $rec(Iter, s)(0)$ to verify the base case, and then proving by induction on n the equality $\widehat{A}(s(n)) = Iter(\widehat{A}(n))$. To complete our proof we just need to show that *Iter* is definable by primitive recursion. This is easy to do with some currying yoga: it is just a matter of unpacking the definitions to check that *Iter* satisfies the defining equation $Iter = \widehat{Iter_0}; [\mathbb{N}{\to}p_2]$, where $\widehat{Iter_0}$ is the currying of $Iter_0$, and where $Iter_0$ is itself defined by primitive recursion as $Iter_0 = prec((p_1, \_(\_)), (id_{[\mathbb{N}\to\mathbb{N}]}, \_(1)))$, where $(p_1, \_(\_)) : [\mathbb{N}{\to}\mathbb{N}] \times \mathbb{N} \longrightarrow [\mathbb{N}{\to}\mathbb{N}] \times \mathbb{N} : (f, n) \mapsto (f, f(n))$, and $(id_{[\mathbb{N}\to\mathbb{N}]}, \_(1)) : [\mathbb{N}{\to}\mathbb{N}] \longrightarrow [\mathbb{N}{\to}\mathbb{N}] \times \mathbb{N} : f \mapsto (f, f(1))$. □

The moral of this story is that, by allowing "primitive recursion on higher types," that is, by using not only cartesian products of $\mathbb{N}$ but also function sets like $[\mathbb{N}{\to}\mathbb{N}]$, $[[\mathbb{N}{\to}\mathbb{N}]^2{\to}[\mathbb{N}^3{\to}\mathbb{N}]]$, and so on, as types in our function definitions, we can define a strictly bigger class of computable functions than those definable using only product types. This kind of recursion on higher types is what higher-order functional programming languages are good at, so that we can define not only recursive functions, but also recursive "functionals" like *Iter*, that is, recursive functions that take other functions as arguments. They are what Gödel called *primitive recursive functionals of finite type* (for those generated from 0 and s) in his famous Dialectica paper. Gödel's Dialectica ideas and subsequent work on them are surveyed in [2], where it is also shown that the *total* computable functions $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ definable by primitive recursive functionals of finite type form an amazingly general class. However, because of the halting problem for Turing machines, we know that some Turing-computable functions on the natural numbers are not total but only *partial*; and Turing computability is of course co-extensive with computability in a general-purpose programming language supporting (fully general) recursive function definitions. §9.5 sketches how the unique extensional partial function associated to a recursive function definition can be defined by *fixpoint semantics*. Furthermore, a very general method to define total computable functions that can handle Ackermann's function and much more without having to go up to higher types will be presented in §14.

7.3 The Peano Axioms

The Peano axioms are due to Giuseppe Peano and axiomatize the natural numbers as the *existence* of a set $\mathbb{N}$ having an element $0 \in \mathbb{N}$ and a function $s : \mathbb{N} \longrightarrow \mathbb{N}$ such that:

1. $(\forall n \in \mathbb{N})\ 0 \neq s(n)$
2. $s : \mathbb{N} \longrightarrow \mathbb{N}$ is injective, and
3. (Peano Induction) $(\forall T \subseteq \mathbb{N})\ ((0 \in T\ \wedge\ ((\forall x \in T)\ s(x) \in T))\ \Rightarrow T = \mathbb{N})$.

Note that, exactly as in the case of the *Peano-Lawvere* axiom, the Peano axioms give us an *abstract* characterization of the natural numbers. That is, they do not tell us anything about the particular way in which numbers are *represented* in the set $\mathbb{N}$.

We have already seen in Theorem 2 that the von Neumann representation of the natural numbers, guaranteed to exist as a set by the (*Inf*) axiom, satisfies the third axiom of Peano Induction. It is relatively easy, but somewhat tedious, to show that it satisfies also the first two Peano axioms (see, e.g., [24]).

However, proving that the Peano axioms hold for the von Neumann representation of the naturals is in a sense *proving too little:* we should prove that the Peano axioms hold for *any* representation of the natural numbers satisfying the *Peano-Lawvere* axiom (part of this proof is already contained in Exercise 49). This, in turn, will imply it for the von Neumann representation thanks to Theorem 5.

In fact, we can prove more than that. As implicit already in [31], and shown in Prop. 17 of [42] as a consequence of a more general result that applies not only to the natural numbers but to any so-called initial algebra,[1] we have in fact an *equivalence*:

$$\textit{Peano} \quad \Leftrightarrow \quad \textit{Peano-Lawvere}$$

in the precise sense that a set $\mathbb{N}$ together with an element $0 \in \mathbb{N}$ and a function $s : \mathbb{N} \longrightarrow \mathbb{N}$ satisfies the *Peano* axioms iff it satisfies the properties stated for $\mathbb{N}$, $0 \in \mathbb{N}$ and $s : \mathbb{N} \longrightarrow \mathbb{N}$ in the *Peano-Lawvere* axiom.

Exercise 51 *Give two set theory formulas expressing precisely the Peano axioms (as a single formula) and the Peano-Lawvere axiom (again, as a single formula).*

[1] The *Peano-Lawvere* axiom can be trivially reformulated in algebraic terms as the claim that an initial algebra exists for the class of algebras whose only operations are a constant 0 and a unary operation s.

Chapter 8

Case Study: The Peano Language

Chapter 9

Binary Relations on a Set

The case of binary relations whose domain and codomain coincide, that is, relations of the form $R \in \mathcal{P}(A \times A)$ is so important as to deserve a treatment of its own. Of course, *any* relation can be viewed as a binary relation on a single set, since for any sets A and B, if $R \in \mathcal{P}(A \times B)$, then $R \in \mathcal{P}((A \cup B) \times (A \cup B))$ (see Exercise 31-(5)). That is, the domain and codomain of a relation R is not uniquely determined by the set R. To avoid this kind of ambiguity, a binary relation on a set A should be specified as a *pair* (A, R), with A the given set, and $R \in \mathcal{P}(A \times A)$.

9.1 Directed and Undirected Graphs

The first obvious thing to observe is that a *directed graph* with set of nodes N and a binary relation $G \in \mathcal{P}(N \times N)$ are *the same thing!* We just use a different terminology and notation to skin the same cat. For example, we now call a pair $(a, b) \in G$ a *directed edge* in the graph G, and then use the graphical notation:

$$a \longrightarrow b \qquad\qquad c \circlearrowleft$$

for, respectively, a pair (a, b) with $a \neq b$, and a pair (c, c). To make the set N of nodes unambiguous, we then specify the graph as the pair (N, G). One important notion in a directed graph is that of a *path*, which is defined as a finite sequence of edges $a_0 \to a_1 \to a_2 \ldots a_{n-1} \to a_n$ with $n \geq 1$. A directed graph (N, G). is *strongly connected* iff for each $a, b \in N$, if $a \neq b$ then there is a path from a to b. (N, G) is *acyclic* iff no node $a \in N$ has a path from a to a. A directed graph (N, G) is *connected* iff the directed graph $(N, G \cup G^{-1})$ is strongly connected. The identity between graphs and binary relations on a set is very useful, since any notion or construction on relations can be interpreted geometrically as a notion or construction on directed graphs. For example:

Exercise 52 *Prove the following:*

1. *$G \in \mathcal{P}(N \times N)$ is a total relation iff G is a directed graph where each node has at least one edge coming out of it.*
2. *$f \in [N \to N]$ iff f is a directed graph where each node has exactly one edge coming out of it.*
3. *$f \in [N \to N]$ is* surjective *iff f is a directed graph where each node has exactly one edge coming out of it and at least one edge coming into it.*
4. *$f \in [N \to N]$ is* injective *iff f is a directed graph where each node has exactly one edge coming out of it and at most one edge coming into it.*
5. *$f \in [N \to N]$ is* bijective *iff f is a directed graph where each node has exactly one edge coming out of it and exactly one edge coming into it.*
6. *A directed graph $G \in \mathcal{P}(N \times N)$ is* strongly connected *iff for each $a, b \in N$ with $a \neq b$ there exists an $n \geq 1$ such that $(a, b) \in G^n$, where we define[1] $G^1 = G$, and $G^{n+1} = G; G^n$.*

[1]Note that this definition is based on simple recursion. Note also the potential ambiguity with the totally different use of the notation G^n for the n-th cartesian product of G with itself. Here, of course, what the notation G^n denotes is the n-th *composition* $G; .^n.; G$ of G with itself.

7. *A directed graph $G \in \mathcal{P}(N \times N)$ is* connected *iff for each $a, b \in N$ with $a \neq b$ there exists an $n \geq 1$ such that $(a, b) \in (G \cup G^{-1})^n$.*
8. *A directed graph $G \in \mathcal{P}(N \times N)$ is* acyclic *iff for each $n \geq 1$, $G^n \cap id_N = \emptyset$.*

What is an *undirected* graph U on a set N of nodes? Of course it is just a subset U of the form $U \subseteq N \otimes N$, that is, an element $U \in \mathcal{P}(N \otimes N)$. In other words, an undirected graph with nodes N is *exactly* a set of unordered pairs of elements of N. Again, to make the set N of nodes unambiguous, we specify an undirected graph as a pair (N, U), with $U \in \mathcal{P}(N \otimes N)$.

This is part of a broader picture. The same way that a relation R from A to B is an element $R \in \mathcal{P}(A \times B)$, we can consider instead elements $U \in \mathcal{P}(A \otimes B)$. Let us call such a U a *linking* between A and B, since it links elements of A and elements of B. An undirected graph on nodes N is then a linking from N to N. Each particular *link* in a linking $U \in \mathcal{P}(A \otimes B)$ is exactly an unordered pair $\{a, b\}$, with $a \in A$ and $b \in B$, which we represent graphically as:

$$a \text{ — } b \qquad\qquad c \bigcirc$$

when, respectively, $a \neq b$, and when the pair is of the form $\{c, c\} = \{c\}$ for some $c \in A \cap B$.

We can then introduce some useful notation for linkings. We write $U : A \Longleftrightarrow B$ as an abbreviation for $U \in \mathcal{P}(A \otimes B)$. Note that the bidirectionality of the double arrow is very fitting here, since $A \otimes B = B \otimes A$ (see Exercise 14), and therefore $\mathcal{P}(A \otimes B) = \mathcal{P}(B \otimes A)$. That is, U is a linking from A to B iff it is a linking from B to A. Given linkings $U : A \Longleftrightarrow B$ and $V : B \Longleftrightarrow C$, we can then define their *composition* $U; V : A \Longleftrightarrow C$ as the linking

$$U; V = \{\{a, c\} \in A \otimes C \mid (\exists b \in B)\ \{a, b\} \in U \ \wedge\ \{b, c\} \in V\}.$$

Note that, in particular, for any set A we have the linking $\widehat{id}_A = \{\{a\} \in A \otimes A \mid a \in A\}$, which we call the *identity linking* on A. We say that an undirected graph (N, U) is *connected* iff for each $a, b \in N$ with $a \neq b$ there exists an $n \geq 1$ such that $(a, b) \in U^n$, where we define $U^1 = U$, and $U^{n+1} = U; U^n$.

Note that every relation, by forgetting about directionality, determines a corresponding linking. This is because for any two sets A and B we have a surjective function

$$und : A \times B \ \twoheadrightarrow A \otimes B : (x, y) \mapsto \{x, y\}$$

which has the equivalent, intensional definition $\lambda x.\ \bigcup x$, since $und(a, b) = und(\{\{a\}, \{a, b\}\}) = \bigcup\{\{a\}, \{a, b\}\} = \{a, b\}$. Note that *und* induces a surjective function

$$und[\_] : \mathcal{P}(A \times B) \ \twoheadrightarrow \mathcal{P}(A \otimes B) : R \mapsto und[R].$$

In particular, for $G \in \mathcal{P}(N \times N)$ a directed graph on nodes N, $und[G]$ is the undirected graph obtained by forgetting the directionality of the edges $a \longrightarrow b$ (resp., $c \circlearrowleft$) in G and turning them into bidirectional links a — b (resp., $c \bigcirc$).

We call a binary relation $R \in \mathcal{P}(A \times A)$ on a set A *symmetric* iff $R = R \cup R^{-1}$. Given any relation $R \in \mathcal{P}(A \times A)$, the smallest symmetric relation containing R is precisely $R \cup R^{-1}$, which is called the *symmetric closure* of R. Note that $und[R] = und[R \cup R^{-1}]$. That is, a relation and its symmetric closure determine the same linking.

Exercise 53 *Give a necessary and sufficient condition φ on A and B, so that the surjective function $und : A \times B \ \twoheadrightarrow A \otimes B$ is bijective iff φ holds. Note that if φ holds, then the surjective function $und[\_] : \mathcal{P}(A \times B) \ \twoheadrightarrow \mathcal{P}(A \otimes B)$ is also bijective, which means that we can then uniquely recover the relation R from its corresponding linking $und(R)$. Give examples of sets A and B such that und is not bijective, so that in general we cannot uniquely recover R from $und(R)$.*

Exercise 54 *Prove the following:*

1. *Given a linking $U : A \Longleftrightarrow B$, we have the equalities $\widehat{id}_A; U = U$ and $U; \widehat{id}_B = U$.*
2. *Given linkings $U : A \Longleftrightarrow B$ and $V : B \Longleftrightarrow C$, their composition is* commutative*, that is, we have the equality of linkings $U; V = V; U$.*
3. *Give an example of sets A, B, C, D, and linkings $U : A \Longleftrightarrow B$, $V : B \Longleftrightarrow C$, and $W : C \Longleftrightarrow D$, showing that linking composition in general is* not associative*, so that we have $(U; V); W \neq U; (V; W)$.*
4. *Given relations $F : A \Longrightarrow B$ and $G : B \Longrightarrow C$, show that $und[F; G] \subseteq und[F]; und[G]$. Exhibit concrete relations F and G such that $und[F; G] \subset und[F]; und[G]$. Prove that if $F, G \in \mathcal{P}(A \times A)$ are symmetric relations then $und[F; G] = und[F]; und[G]$.*
5. *For any set A we always have $und[id_A] = \widehat{id}_A$.*
6. *A directed graph $G \in \mathcal{P}(N \times N)$ is connected iff its corresponding undirected graph $und[G]$ is connected.*

9.2 Transition Systems and Automata

What is a transition system on a set A of states? Of course, it is *exactly* a binary relation on the set A. So, this is a third, equivalent skinning of the *same* cat. In other words, we have the equality:

$$\textit{Binary Relation} \;=\; \textit{Directed Graph} \;=\; \textit{Transition System}$$

where, depending on what kinds of applications we are most interested in, we adopt a somewhat different notation and terminology. But although the words are different, the underlying concepts *are* the same. Now instead of calling the elements of A nodes, we call them *states*. And instead of using letters like F, G, etc. to denote the binary relation, we define a *transition system* as a pair $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$, where A is its set of *states* and $\rightarrow_{\mathcal{A}} \in \mathcal{P}(A \times A)$ is its *transition relation*. And instead of calling a pair $(a, a') \in \rightarrow_{\mathcal{A}}$ and edge, we call it a *transition*, and write it $a \rightarrow_{\mathcal{A}} a'$, or just $a \rightarrow a'$ when the given $\mathcal{A}$ is understood. So, we have changed the terminological window dressing, but essentially we have not changed *anything*: we are still talking about the same thing.

In a transition system many of the issues we care about are *reachability* issues: given an initial state a, can we reach a state a' by a sequence of system transitions? We also care about *deadlocks*: are there states in which we get stuck and cannot go on to a next state? Futhermore, we care about *termination*: does every sequence of transitions always eventually come to a stop? These are all obvious relational notions disguised under a systems-oriented terminology.

Definition 13 *A binary relation $R \in \mathcal{P}(A \times A)$ is called* reflexive *iff $id_A \subseteq R$. And it is called* irreflexive *iff $id_A \cap R = \varnothing$.*

A binary relation $R \in \mathcal{P}(A \times A)$ is called transitive *iff*

$$(\forall a, a', a'' \in A)\ (((a, a') \in R \ \wedge\ (a', a'') \in R) \Rightarrow (a, a'') \in R)$$

The following, trivial lemma is left as an exercise.

Lemma 6 *Given any binary relation $R \in \mathcal{P}(A \times A)$, the smallest reflexive relation containing it is the relation $R^{=} = R \cup id_A$, which we call its* reflexive closure.

Given any binary relation $R \in \mathcal{P}(A \times A)$, the smallest transitive relation containing it is the relation

$$R^{+} = \bigcup \{R^n \in \mathcal{P}(A \times A) \mid n \in (\mathbb{N} - \{0\})\}$$

which we call its transitive closure *(for the definition of R^n see Exercise 52-(6)).*

Given any binary relation $R \in \mathcal{P}(A \times A)$, the smallest reflexive and transitive relation containing it is the relation $R^{*} = R^{+} \cup id_A$, which we call its reflexive and transitive closure.

Given a transition system $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$, what does it mean to say that a state a' is *reachable* from a state a? It means exactly that $a \rightarrow^{*}_{\mathcal{A}} a'$. And what is a *deadlock state*? It is a state $a \in A$ such that there is no $a' \in A$ with $a \rightarrow_{\mathcal{A}} a'$. And what does it mean to say that $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ is *deadlock-free* (has no deadlock states)? It means exactly that $\rightarrow_{\mathcal{A}}$ is a *total* relation.

The best way to make precise the notion of *termination* of $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ is to explain what it means for $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$, *not* to terminate. This obviously means that there is a function $a : \mathbb{N} \longrightarrow A$ such that for each $n \in \mathbb{N}$ we have $a(n) \rightarrow_{\mathcal{A}} a(n+1)$. We call a function a satisfying this property an *infinite computation* of $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$, and display such infinite computations graphically as follows:

$$a(0) \rightarrow_{\mathcal{A}} a(1) \rightarrow_{\mathcal{A}} a(2) \rightarrow_{\mathcal{A}} a(3)\ \ldots\ a(n) \rightarrow_{\mathcal{A}} a(n+1)\ \ldots$$

We then say that $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ is *terminating* iff it has no infinite computations.

Of course, since *Binary Relation = Directed Graph = Transition System*, we likewise call a binary relation (A, R) (resp., a directed graph (A, G)) *terminating* (or *well-founded*, see §14), iff there is no infinite computation in (A, R) (resp., in (A, G)). With the obvious slight change of notation, this exactly means that there is no function $a : \mathbb{N} \longrightarrow A$ such that for each $n \in \mathbb{N}$ we have $(a(n), a(n+1)) \in R$ (resp., $a(n) \rightarrow a(n+1)$, or $a(n) \circlearrowleft$ if $a(n) = a(n+1)$, is a directed edge in (A, G)).

Automata are just a variant of transition systems in which transitions have labels, which we think of as *inputs* or *events* or *actions* of the system.

Definition 14 *A* labeled transition system *or* automaton *is a triple* $\mathcal{A} = (A, L, \rightarrow_{\mathcal{A}})$*, where* A *is its set of* states*,* L *is its set of* labels *or* inputs*, and* $\rightarrow_{\mathcal{A}} \in \mathcal{P}(A \times L \times A)$ *is its* transition relation*.*

The triples $(a, l, a') \in \rightarrow_{\mathcal{A}}$ *are called the* labeled transitions*, and are denoted* $a \overset{l}{\rightarrow}_{\mathcal{A}} a'$*.*

Note, again, that due to the identity *Directed Graph = Transition System*, the above definition is (up to a trivial change of terminology, changing "state" by "node," and "labeled transition" by "labeled edge") *exactly* the definition of a *labeled graph*. Relation? graph? transition system? They are all the *same!*

Finally, note that the usual description of a (nondeterministic) automaton as a triple $\mathcal{A} = (A, L, \delta_{\mathcal{A}})$, where A is the set of states, L is the set of inputs, and $\delta_{\mathcal{A}} : L \times A \Longrightarrow A$ is the transition relation, is equivalent to the above definition $\mathcal{A} = (A, L, \rightarrow_{\mathcal{A}})$, just by swapping the order of the first two components in the corresponding triple. That is, for the same A and L we can obtain two *equivalent* representations of the *same* automaton $\mathcal{A}$ (one with $\rightarrow_{\mathcal{A}}$, and another with $\delta_{\mathcal{A}}$) by means of the defining equivalence:

$$((l, a), a') \in \delta_{\mathcal{A}} \quad \Leftrightarrow \quad a \overset{l}{\rightarrow}_{\mathcal{A}} a'$$

Of course, a *deterministic automaton* is the special case when $\delta_{\mathcal{A}} : L \times A \Longrightarrow A$ is not just a relation, but a function $\delta_{\mathcal{A}} : L \times A \longrightarrow A$, called the automaton's *transition function*.

9.3 Relation Homomorphisms and Simulations

Given graphs (N, G) and (N', G'), we call a function $f : N \longrightarrow N'$ a *graph homomorphism* from (N, G) to (N', G') iff $(\forall x, y \in N)\ (x, y) \in G \ \Rightarrow\ (f(x), f(y)) \in G'$. We use the notation $f : (N, G) \longrightarrow (N', G')$ as a shorthand for: "f is a graph homomorphism from (N, G) to (N', G')." Note that: (i) id_N is always a graph homomorphism from (N, G) to itself; and (ii) if we have $f : (N, G) \longrightarrow (N', G')$ and $g : (N', G') \longrightarrow (N'', G'')$, then we have $f; g : (N, G) \longrightarrow (N'', G'')$. The notion of graph homomorphism is of course heavily used in combinatorics and graph algorithms (or special cases of it, such as that of a graph isomorphism, on which more below), and is very intuitive: we map nodes of G to nodes of G', and require that each edge in G should be mapped to a corresponding edge in G'.

But we mustn't forget our identity: *Binary Relation = Directed Graph = Transition System*. So, why calling $f : (N, G) \longrightarrow (N', G')$ a graph homomorphism? This terminology is just in the eyes of the beholder. We may as well call it a *relation homomorphism* (since it respects the corresponding relations G and G'), or, alternatively, a *simulation map* between transition sytems, since (changing notation a little), if we have transition systems $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ and $\mathcal{B} = (B, \rightarrow_{\mathcal{B}})$, then a graph homomorphism $f : (A, \rightarrow_{\mathcal{A}}) \longrightarrow (B, \rightarrow_{\mathcal{B}})$ exactly tells us that any transition $a \rightarrow_{\mathcal{A}} a'$ in system $\mathcal{A}$ can always be *simulated* via f by a transition $f(a) \rightarrow_{\mathcal{B}} f(a')$ in system $\mathcal{B}$. Therefore, we again have the *same* notion under different names, that is, the identity:

$$\textit{Relation Homomorphism} \ = \ \textit{Graph Homomorphism} \ = \ \textit{Simulation Map}.$$

Of course, the general idea of a "homomorphism" of any kind is that of a function that "preserves the relevant structure." For relation homomorphisms the relevant structure is that of a graph that, alternatively, can be regarded as a transition system, or just as a relation on a set, and the preservation of structure for $f : (N, G) \longrightarrow (N', G')$ is precisely the condition $(\forall x, y \in N)\ (x, y) \in G \ \Rightarrow\ (f(x), f(y)) \in G'$. In a completely analogous way, we have seen that certain data types, such as products, disjoint unions, and models of the natural numbers, have a relevant "structure," namely, what in computer science are called their "interfaces," that is, the projection functions p_1, p_2 for products, the injection functions i_1, i_2 for disjoint unions, and the zero, 0, and successor function, s, for models of the natural numbers. Implicitly, in Exercises 40 and 48 we were using the adequate notion of homomorphism preserving the relevant structure for abstract products, disjoint unions, and models of the natural numbers in the special case of *isomorphisms*, that is, of bijective homomorphism (for the relevant structure) whose inverse is also a homomorphism (again, for the relevant structure).

For relation homomorphism the definition of isomorphism is obvious. A relation homomorphism $f : (N, G) \longrightarrow (N', G')$ is called a relation *isomorphism* iff f is bijective and f^{-1} is also a relation homomorphism $f^{-1} : (N', G') \longrightarrow (N, G)$. When we take the graph-theoretic point of view this is *exactly* the

well-known notion of *graph isomorphism*, that is, two graphs that are esentially the same graph except for a renaming of their nodes. The graphs $(\{a,b,c\},\{(a,b),(b,c),(c,a)\})$, and $(\{e,f,g\},\{(e,f),(f,g),(g,e)\})$ are isomorphic graphs; and the function $a \mapsto e,\ b \mapsto f,\ c \mapsto g$, and also the function $a \mapsto f,\ b \mapsto g,\ c \mapsto e$, are both examples of graph isomorphisms between them. Therefore, abstractly they are the *same* graph.

Of course, we call a relation homomorphism $f : (N,G) \longrightarrow (N',G')$ *injective*, resp., *surjective*, resp., *bijective* iff the function f is injective, resp., surjective, resp., bijective. Note that, although every relation isomorphism is necessarily bijective, some bijective relation homomorphisms are *not* relation isomorphisms (can you give a simple example?).

A special case of injective relation homomorphism is that of a *subrelation*, that is, we call (N,G) a *subrelation* of (N',G') if $N \subset N'$ and the inclusion map $j_N^{N'}$ is a relation homomorphisms $j_N^{N'} : (N,G) \hookrightarrow (N',G')$. Note that (N,G) is a *subrelation* of (N',G') iff $N \subset N'$ and $G \subset G'$; for this reason we sometimes use the suggestive notation $(N,G) \subseteq (N',G')$ to indicate that (N,G) is a subrelation of (N',G'). Note that, from a graph-theoretic point of view, a subrelation is *exactly* a *subgraph*. Of course, viewed as a transition system, a subrelation $j_N^{N'} : (N,G) \hookrightarrow (N',G')$ is exactly a *transition subsytem*.

We call a relation homomorphism $f : (N,G) \longrightarrow (N',G')$ *full* iff $(\forall x,y \in N)\ (x,y) \in G \Leftrightarrow (f(x),f(y)) \in G'$. Likewise, we call a subrelation $j_N^{N'} : (N,G) \hookrightarrow (N',G')$ a *full subrelation* iff $j_N^{N'}$ is a full relation homomorphism. Note that $j_N^{N'} : (N,G) \hookrightarrow (N',G')$ is a full subrelation iff $G = G' \cap N^2$. Therefore, to indicate that $(N,G) \subseteq (N',G')$ is a full subrelation, we write $G = G' \cap N^2 = G'|_N$, write $(N,G'|_N) \subseteq (N',G')$, and call $G'|_N$ the *restriction*[2] of G' to N. Graph-theoretically, a full subrelation $(N,G'|_N) \subseteq (N',G')$ is exactly a full subgraph, that is, we restrict the nodes to N, but keep all the edges from G' that begin and end in nodes from N.

The above notion of relation homomorphism can be generalized in three ways. The first generalization is to extend this notion to a notion of *homomorphism of labeled graphs*, or, equivalently, *labeld simulation map of labeled transition systems* or *automata*. Given that a labeled graph is exactly the same thing as labeled transition system, to avoid unnecessary repetitions I only define the notion using the transition system terminology. Given labeled transition systems $\mathcal{A} = (A,L,\rightarrow_{\mathcal{A}})$ and $\mathcal{B} = (B,L,\rightarrow_{\mathcal{B}})$ having the same set L of labels, a *labeled simulation map* from $\mathcal{A}$ to $\mathcal{B}$, denoted $f : \mathcal{A} \longrightarrow \mathcal{B}$, is a function $f : A \longrightarrow B$ such that whenever we have a labeled transition $a \overset{l}{\rightarrow}_{\mathcal{A}} a'$ in $\mathcal{A}$, then $f(a) \overset{l}{\rightarrow}_{\mathcal{B}} f(a')$ is a labeled transition in $\mathcal{B}$. That is, $\mathcal{B}$ can copycat anything $\mathcal{A}$ can do with the *same* labels.

A second generalization considers relation homomorphisms $H : (N,G) \Longrightarrow (N',G')$, where now H is not a function but a relation. Again, the three viewpoints of relation, graph, and transition system are equivalent. I give the definition using the transition system terminology, where it is most often used and is called a *simulation relation*. In more neutral terms it could be called a *non-deterministic relation homomorphism*. Given transition systems $\mathcal{A} = (A,\rightarrow_{\mathcal{A}})$ and $\mathcal{B} = (B,\rightarrow_{\mathcal{B}})$, a *simulation relation* from $\mathcal{A}$ to $\mathcal{B}$, denoted $H : \mathcal{A} \Longrightarrow \mathcal{B}$, is a relation $H : A \Longrightarrow B$ such that whenever we have aHb and a transition $a \rightarrow_{\mathcal{A}} a'$ in $\mathcal{A}$, then there is a $b' \in B$ such that $b \rightarrow_{\mathcal{B}} b'$ is a transition in $\mathcal{B}$ and $a'Hb'$. We call $H : \mathcal{A} \Longrightarrow \mathcal{B}$ a *bisimulation* iff $H^{-1} : \mathcal{B} \Longrightarrow \mathcal{A}$ is also a simulation relation. That is, $\mathcal{B}$ can copycat $\mathcal{A}$, *and* $\mathcal{A}$ can copycat $\mathcal{B}$, so that $\mathcal{A}$ and $\mathcal{B}$ are *behavioraly equivalent*.

A third generalization combines the previous two into the notion of a *labeled simulation relation*. Given labeled transition systems $\mathcal{A} = (A,L,\rightarrow_{\mathcal{A}})$ and $\mathcal{B} = (B,L,\rightarrow_{\mathcal{B}})$ having the same set L of labels, a *labeled simulation relation* from $\mathcal{A}$ to $\mathcal{B}$, denoted $H : \mathcal{A} \Longrightarrow \mathcal{B}$, is a relation $H : A \Longrightarrow B$ such that whenever we have a labeled transition $a \overset{l}{\rightarrow}_{\mathcal{A}} a'$ in $\mathcal{A}$ and aHb, then there is a $b' \in B$ and a labeled transition $b \overset{l}{\rightarrow}_{\mathcal{A}} b'$ in $\mathcal{B}$ such that $a'Hb'$. We call $H : \mathcal{A} \Longrightarrow \mathcal{B}$ a *labeled bisimulation* iff $H^{-1} : \mathcal{B} \Longrightarrow \mathcal{A}$ is also a labeled simulation relation. That is, $\mathcal{B}$ can copycat $\mathcal{A}$ with the same labels, *and* $\mathcal{A}$ can copycat $\mathcal{B}$ also with the same labels, so that $\mathcal{A}$ and $\mathcal{B}$ are *behavioraly equivalent* in an even stronger sense.

Exercise 55 *Prove that $f : (N,G) \longrightarrow (N',G')$ is a relation isomorphism iff f is a bijective and full relation homomorphism.*

Exercise 56 *(Pulling back binary relations). Let (B,G) be a binary relation and let $f : A \longrightarrow B$ be a function. Prove that: (i) $f : (A, f^{-1}(G)) \longrightarrow (B,G)$ is a relation homomorphism, where, by definition, $a\, f^{-1}(G)\, a' \Leftrightarrow f(a)\, G\, f(a')$; and*

[2]Note that this notion of restriction is similar to, but in general different from, the notion of restriction of a function or a relation to a *subdomain* defined in Exercise 32. The difference is that for $R : A \Rightarrow B$, and $A' \subseteq A$, $R\restriction_{A'} = R \cap (A' \times B)$. Even when $A = B$, in general this yields a different notion of restriction than $R|_{A'} = R \cap A'^2$. For this reason, the two different notions: $R\restriction_{A'}$, and, when $A = B$, $R|_{A'}$, are each given a *different* notation.

(ii) $f : (A, R) \longrightarrow (B, G)$ is a relation homomorphism iff $R \subseteq f^{-1}(G)$.

Exercise 57 *(Peano-Lawvere is an "Infinity Axiom"). Prove that any set $\mathbb{N}$ with associated functions $\widehat{0} : 1 \longrightarrow \mathbb{N}$ and $s : \mathbb{N} \longrightarrow \mathbb{N}$ that satisfies the Peano-Lawvere axiom must be infinite. This tells us that we can use the Peano-Lawvere axiom as an alternative to (Inf) to ensure the existence of infinite sets in out set theory. (Hint: Use Exercise 52 (2) to view $(\mathbb{N}, s)$ as a directed graph.)*

Exercise 58 *Prove the following:*

1. *If $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ is a transition system, then id_A is a simulation map $id_A : \mathcal{A} \longrightarrow \mathcal{A}$. And if $f : \mathcal{A} \longrightarrow \mathcal{B}$ and $g : \mathcal{B} \longrightarrow C$ are simulation maps, then $f; g : \mathcal{A} \longrightarrow C$ is also a simulation map.*
2. *Prove the exact same properties as in (1) changing:*
   - *"transition system" to "labeled transition system," and "simulation map" to "labeled simulation map" everywhere*
   - *"simulation map" to "simulation relation" everywhere*
   - *"transition system" to "labeled transition system," and "simulation map" to "labeled simulation relation" everywhere.*
3. *If $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ and $\mathcal{B} = (B, \rightarrow_{\mathcal{B}})$ are transition systems and $\mathcal{H} \subseteq \mathcal{P}(A \times B)$ is a set of binary relations such that each $H \in \mathcal{H}$ is a simulation relation (resp. a bisimulation relation) $H : \mathcal{A} \Longrightarrow \mathcal{B}$, then $\bigcup \mathcal{H}$ is also a simulation relation (resp. a bisimulation relation) $\bigcup \mathcal{H} : \mathcal{A} \Longrightarrow \mathcal{B}$. Prove the same changing "transition system" to "labeled transition system," and "simulation relation" to "labeled simulation relation" everywhere.*
4. *For any transition systems $\mathcal{A} = (A, \rightarrow_{\mathcal{A}})$ and $\mathcal{B} = (B, \rightarrow_{\mathcal{B}})$ there is a "biggest possible" simulation (resp. bisimulation) between them, that is, a simulation relation $max : \mathcal{A} \Longrightarrow \mathcal{B}$ (resp. a bisimulation relation $max.bis : \mathcal{A} \Longrightarrow \mathcal{B}$) such that for any other simulation (resp. bisimulation) relation $H : \mathcal{A} \Longrightarrow \mathcal{B}$ we always have $H \subseteq max$ (resp. $H \subseteq max.bis$). Prove the same changing "transition system" to "labeled transition system," "simulation relation" to "labeled simulation relation," and "bisimulation relation" to "labeled bisimulation relation" everywhere.*

9.4 Orders

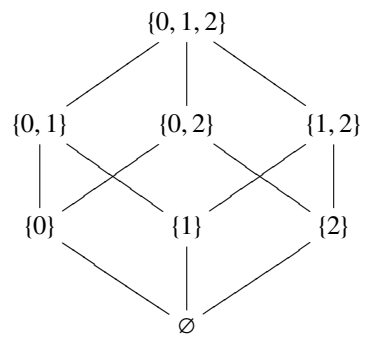
A (strict) *order* on a set A is a relation $< \in \mathcal{P}(A \times A)$ that is both *transitive* and *irreflexive*. We read $a < a'$ as "a is less than a'," or "a is smaller than a'." The inverse relation $<^{-1}$ is denoted $>$, and we read $a > a'$ as "a is greater than a'." An *ordered set* is a set with an order on it, that is, a pair $(A, <)$, with $<$ a (strict) order.

Note the pleasing fact that if $<$ is transitive and irreflexive, then $>$ is also transitive and irreflexive. Therefore if we adopt $>$ as our "less than" relation, we get another order on A, namely $(A, >)$, which is called the *opposite*, or *inverse*, or *dual* order relation. For example, we can order the natural numbers in either the usual ascending order, or in their opposite, descending order.

Examples of ordered sets are everywhere. The $<$ relation on the natural numbers $\mathbb{N}$, the integers $\mathbb{Z}$, the rationals $\mathbb{Q}$, and the reals $\mathbb{R}$, make all these sets ordered sets. Likewise, the strict containment relation $\subset$ makes any powerset $\mathcal{P}(A)$ into an ordered set $(\mathcal{P}(A), \subset)$. A different order on $\mathbb{N} - \{0, 1\}$ is given by *divisibility*, where we write $n|m$, read "n (strictly) divides m," to abbreviate the formula $(\exists k \in \mathbb{N} - \{0, 1\})\ k \cdot n = m$. Then $(\mathbb{N} - \{0, 1\}, \_|\_)$ is an order.

Exploiting again the equality *Directed Graph = Relation*, what is an ordered set when viewed as a graph? It is exactly a directed *transitive and acyclic* graph. So talk about an ordered set and talk about a directed transitive and acyclic graph is just talk about the *same* thing.

Of course, the transitive closure G^+ of any directed acyclic graph G is a directed transitive and acyclic graph. This is exploited pictorially in the so-called *Hasse diagram* of an ordered set, so that the ordered set is pictured as a directed acyclic graph (with the bigger nodes depicted above the smaller nodes without explicitly drawing the arrowheads), and without explicitly drawing the remaining edges in the transitive closure, although they are understood to also be there. For example, the Hasse diagram of the ordered set $(\mathcal{P}(3), \subset)$ is the following directed acyclic graph:



Given an ordered set $(A, <)$, the reflexive closure $<^{=}$ of its (strict) order relation is always denoted $\leq$. We then read $a \leq a'$ as "a is less than or equal to a'," or "a is smaller than or equal to a'." Of course, the inverse relation $\leq^{-1}$ is always denoted $\geq$, and we read $a \geq a'$ as "a is greater than or equal to a'." Note that, since $<$ is irreflexive, the mappings: (i) $< \mapsto \leq$, with $\leq = (< \cup id_A)$, and (ii) $\leq \mapsto <$, with $< = (\leq - id_A)$, are inverse to each other, so that we can get the strict order $<$ from its nonstrict version $\leq$, and *vice versa*. So we can equivalently talk of an ordered set as either the pair $(A, <)$, or the pair $(A, \leq)$, specifying either the strict order $<$, or its reflexive closure $\leq$.

Sometimes an ordered set $(A, <)$ is called a *partially ordered* set, or a *poset* for short. This is to emphasize that elements in an ordered set may be *incomparable*. For example, in the ordered set $(\mathcal{P}(3), \subset)$, the singleton sets $\{0\}$ and $\{1\}$ are incomparable, since $\{0\} \not\subset \{1\}$ and $\{1\} \not\subset \{0\}$. Instead, any two different numbers in $(\mathbb{N}, <)$, or $(\mathbb{Z}, <)$, or $(\mathbb{Q}, <)$, or $(\mathbb{R}, <)$ can always be compared. An ordered set where elements can always be compared is called a *totally ordered set* (also called a *lineally ordered set*, or a *chain*). The precise definition of a poset $(A, <)$ being totally ordered is that it satisfies the formula

$$(\forall x, y \in A)\ (x < y \ \vee\ x = y \ \vee\ x > y).$$

(or, equivalently, it satisfies the formula $(\forall x, y \in A)\ (x \leq y \ \vee\ x \geq y)$.) For example, $(\mathbb{N}, <)$, $(\mathbb{Z}, <)$, $(\mathbb{Q}, <)$, and $(\mathbb{R}, <)$ are all total orders. So are $(\mathcal{P}(\varnothing), \subset)$ and $(\mathcal{P}(1), \subset)$, the only two powersets where set containment is a total order. The alternate description of a total order as a "linear" order, or a "chain," comes from its pictorial representation as a graph, since in such a representation all the elements are arranged on a single vertical line.

One consequence of the partial nature of non-linear orders is that, in general, a partially ordered set may have zero, one, or more than one element such that there is no other element bigger than such an element. For example, in the poset $(\mathcal{P}(3) - \{3\}, \subset)$, the elements $\{0, 1\}$, $\{0, 2\}$, and $\{1, 2\}$, are exactly those elements such that there is no other element bigger than any of them, since $3 = \{0, 1, 2\}$ has been removed. Such elements, if they exist, are called the *maximal* elements of the poset. More precisely, an element $a \in A$ is called a *maximal element* in a poset $(A, <)$ iff it satisfies the formula $(\forall x \in A)\neg(x > a)$; equivalently, a is maximal iff $<[\{a\}] = \varnothing$.

One very stupid and unfair, yet very common, fallacy in human affairs comes from the deeply confused idea that if there is an order ranking human beings under some criteria (already a questionable matter), that order must surely be linear. Unfortunately, the thought that an order *could* be partial does not even enter into many confused minds. This leads to unrestricted talk about *the best* person in some respect or another. But such talk is often both false and unfair.

First of all there is the problem of determining how "good" in relation to some criteria people are, which often may involve a lot of subjective perceptions. For the sake of argument, let us grant that a fair, objective evaluation may be possible in some cases. Even under such ideal circumstances, and assuming we can honestly conclude that an individual is *unsurpassed* in the qualities being measured, which exactly means that he/she is a *maximal* element under the agreed criteria of comparison (for example, a better parent, a better researcher, a better employee, a better student), it is in general *fallacious* to conclude that there is a *unique unsurpassed* individual, that is, that there is such a thing as "the best" parent, researcher, employee, student, or whatever. A little applied set theory can go a long way in seeing through the falsity of such deeply unfair, yet very common, social practices.

Of course, the maximal elements of the poset $(A, >)$ are called the *minimal* elements of the poset $(A, <)$. That is, an element $a \in A$ is called a *minimal element* in a poset $(A, <)$ iff it satisfies the formula $(\forall x \in$

$A)\neg(x < a)$; equivalently, a is minimal iff $>[\{a\}] = \emptyset$. For example, the minimal elements of the poset $(\mathcal{P}(3) - \{\emptyset\}, \subset)$ are exactly $\{0\}$, $\{1\}$, and $\{2\}$. And of course the minimal elements of $(\mathcal{P}(3) - \{3\}, \supset)$ are exactly $\{0, 1\}$, $\{0, 2\}$, and $\{1, 2\}$.

Given posets $(A, \leq_A)$ and $(B, \leq_B)$, we call a function $f : A \longrightarrow B$ *monotonic* (or *monotone*) from $(A, \leq_A)$ to $(B, \leq_B)$ iff $(\forall a, a' \in A)\ a \leq_A a' \ \Rightarrow\ f(a) \leq_B f(a')$. We use the notation $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ as a shorthand for: "f is a monotonic function from $(A, \leq_A)$ to $(B, \leq_B)$." Have we seen this guy before? Of course! This is just our good old friend Mr. Relation Homomorphism, a.k.a. Mr. Graph Homomorphism, a.k.a. Mr. Simulation Map, disguised under yet another alias! That is, a monotonic function $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ is *exactly* a relation homomorphism $f : (A, \leq_A) \longrightarrow (B, \leq_B)$.

We call a monotonic function $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ *strictly monotonic* iff in addition it satisfies that $(\forall a, a' \in A)\ a <_A a' \ \Rightarrow\ f(a) <_B f(a')$. That is, a strictly monotonic function $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ is *exactly* a relation homomorphism $f : (A, <_A) \longrightarrow (B, <_B)$. Of course, as is more generally the case for relation homomorphisms, we also have that: (i) id_A is always a monotonic (and also strictly monotonic) function from $(A, \leq_A)$ to itself; and (ii) if $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ and $g : (B, \leq_B) \longrightarrow (C, \leq_C)$ are monotonic (resp., strictly monotonic), then so is $f; g$.

For example, the function $\lambda x \in \mathbb{N}.\ x^2 \in \mathbb{N}$ is a strictly monotonic function from $(\mathbb{N}, \leq)$ to itself. And the function $\lambda(x, y) \in \mathbb{N}^2.\ max(x, y) \in \mathbb{N}$, where $max(x, y)$ denotes the maximum of numbers x and y in the order $(\mathbb{N}, \leq)$, is a monotonic but *not* strictly monotonic function from $(\mathbb{N} \times \mathbb{N}, \leq_{\mathbb{N}\times\mathbb{N}})$ to $(\mathbb{N}, \leq)$ (see Exercise 61 for the definition of $(\mathbb{N} \times \mathbb{N}, \leq_{\mathbb{N}\times\mathbb{N}})$).

By definition, a *poset isomorphism* $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ is exactly a relation isomorphism. We call $(X, \leq')$ a *subposet* of $(A, \leq)$ iff $(X, \leq') \subseteq (A, \leq)$ is a subrelation. Similarly a subposet of the form $(X, \leq|_X) \subseteq (A, \leq)$ is called a *full subposet* of $(A, \leq)$. The same definitions apply replacing $\leq$ by $<$ everywhere; for example, a subposet $(X, <') \subseteq (A, <)$ is exactly a subrelation.

Exercise 59 *Call a relation $R \subseteq A \times A$* asymmetric *iff $R \cap R^{-1} = \emptyset$. Call a relation $R \subseteq A \times A$* antisymmetric *iff $R \cap R^{-1} \subseteq id_A$. Prove that the following are equivalent for $R \subseteq A \times A$ a relation:*

- *R is irreflexive and transitive*
- *R is asymmetric and transitive.*

Prove also that the following are equivalent for $R \subseteq A \times A$ an irreflexive *relation:*

- *R is transitive*
- *R is asymmetric and transitive*
- *$R \cup id_A$ is reflexive, antisymmetric and transitive.*

Likewise, prove that the following are equivalent for $R \subseteq A \times A$ a reflexive *relation:*

- *R is antisymmetic and transitive*
- *$R - id_A$ is asymmetric and transitive*
- *$R - id_A$ is irreflexive and transitive.*

Therefore, it is equivalent *to define an ordered set either: (i) as a pair $(A, <)$, with the (strict) order $<$ irreflexive and transitive; or (ii) as a pair $(A, <)$, with $<$ asymmetic and transitive; or (iii) as a pair $(A, \leq)$ with the (nonstrict) order $\leq$ reflexive, transitive and antisymmetric, since, given $<$, we can define $\leq\ = (< \cup id_A)$, and given $\leq$, we can define $<\ = (\leq - id_A)$.*

Exercise 60 *Prove the following:*

1. *$f : (A, \leq_A) \longrightarrow (B, \leq_B)$ is a poset isomorphism iff $f : (A, <_A) \longrightarrow (B, <_B)$ is a strict poset isomorphism.*
2. *If $(A, <_A)$ and $(B, <_B)$ are chains, then a strictly monotonic function $f : (A, <_A) \longrightarrow (B, <_B)$ is a poset isomorphism iff f is surjective.*

Exercise 61 *Given posets $(A, \leq_A)$ and $(B, \leq_B)$, define on the cartesian product $A \times B$ the relation $\leq_{A\times B}$, called the* product order*, by the equivalence*

$$(a, b) \leq_{A\times B} (a', b') \quad \Leftrightarrow \quad (a \leq_A a' \ \wedge\ b \leq_B b')$$

Prove that $\leq_{A\times B}$ is reflexive, transitive, and antisymmetric, and therefore (by Exercise 59) $(A \times B, \leq_{A\times B})$ is a poset.

Show that $(A\times B, \leq_{A\times B})$ is "almost never" a total order, by giving a (in fact quite restrictive) necessary and sufficient condition involving the cardinalities of A and B, and the orders $(A, \leq_A)$ and $(B, \leq_B)$, so that $(A\times B, \leq_{A\times B})$ is a total order iff your condition holds.

State and prove the analogue of Exercise 38 for posets, replacing everywhere the word "set" by "poset," and the word "function" by "monotonic function,"

Can you dualize *all this? Give a definition of* disjoint union of posets $(A, \leq_A) \oplus (B, \leq_B)$*, and show that it is the* right *definition by proving that it satisfies the properties in the analogue of Exercise 39 for posets, replacing everywhere the word set by "poset," and the word "function" by "monotonic function."*

Exercise 62 *Given posets $(A, <_A)$ and $(B, <_B)$, define on the cartesian product $A \times B$ the relation $<_{Lex(A,B)}$, called the* lexicographic order*, by the equivalence*

$$(a, b) <_{Lex(A,B)} (a', b') \quad \Leftrightarrow \quad (a <_A a' \ \vee \ (a = a' \ \wedge \ b <_B b'))$$

Prove that $<_{Lex(A,B)}$ is an order relation, and therefore $(A \times B, <_{Lex(A,B)})$ is a poset.

Prove that if $(A, <_A)$ and $(B, <_B)$ are total orders, then $(A \times B, <_{Lex(A,B)})$ is also a total order.

Exercise 63 *Consider the following statement:*

> *If the set of maximal elements of a poset $(A, <)$ is nonempty, then for each $x \in A$ we can choose a maximal element m_x of $(A, <)$ such that $m_x \geq x$.*

Is this statement true or false? The way to answer any such question is to either give a proof of the statement or to give a counterexample*, that is, an example where the statement fails.*

Exercise 64 *(Embeddings, and Representing a Poset in its Powerset). A monotonic function $f : (A, \leq_A) \longrightarrow (B, \leq_B)$ is called a* monotonic embedding *iff it is a full relation homomorphism. Prove the following:*

1. *A monotonic embedding is a strictly monotonic function and is injective.*
2. *For any poset $(A, \leq)$, the function $\geq[\{\_\}] = \lambda a \in A. \ \geq[\{a\}] \in \mathcal{P}(A)$ defines a monotonic embedding from $(A, \leq)$ to $(\mathcal{P}(A), \subseteq)$. In particular, the function $\geq[\{\_\}]$ defines a poset isomorphism $(A, \leq) \cong (\geq[\{\_\}][A], \subseteq)$. This is a way of "faithfully representing" any poset $(A, \leq)$ inside its powerset poset $(\mathcal{P}(A), \subseteq)$.*
3. *Given a poset $(A, \leq)$, the function $>[\{\_\}] = \lambda a \in A. \ >[\{a\}] \in \mathcal{P}(A)$ is a strictly monotonic function from $(A, \leq)$ to $(\mathcal{P}(A), \subseteq)$. Show that in general this function is not injective (therefore, is not a monotonic embedding either). Show that, however, if $(A, \leq)$ is a chain, then the function $>[\{\_\}]$ is a monotonic embedding and defines a poset isomorphism $(A, \leq) \cong (>[\{\_\}][A], \subseteq)$, providing yet another way of "faithfully representing" a chain $(A, \leq)$ inside its powerset poset $(\mathcal{P}(A), \subseteq)$.*

9.5 Sups and Infs, Complete Posets, Lattices, and Fixpoints

Given a poset $(A, \leq)$ and a subset $X \subseteq A$, we define the set $ubs(X)$ of *upper bounds* of X in $(A, \leq)$ as the set $ubs(X) = \{x \in A \mid (\forall y \in X) \ x \geq y\}$. Dually, we define the set $lbs(X)$ of *lower bounds* of X in $(A, \leq)$ as the set of upper bounds of X in $(A, \geq)$. That is, $lbs(X) = \{x \in A \mid (\forall y \in X) \ x \leq y\}$. For example, in the poset $(\mathcal{P}(3), \subseteq)$ for $X = \{\{0\}, \{2\}\}$ we have $ubs(X) = \{\{0, 2\}, \{0, 1, 2\}\}$, and $lbs(X) = \{\emptyset\}$.

Given a poset $(A, \leq)$ and a subset $X \subseteq A$, we say that X has a *least upper bound* (or *lub*, or *sup*) in $(A, \leq)$ iff $ubs(X) \neq \emptyset$ and there exists an element of $ubs(X)$, denoted $\bigvee X \in ubs(X)$, such that $(\forall y \in ubs(X)) \ \bigvee X \leq y$. Of course, $\bigvee X$, if it exists, is unique, since it is the *smallest* upper bound of X. Dually, we say that X has a *greatest lower bound* (or *glb*, or *inf*) in $(A, \leq)$ iff X has a least upper bound in $(A, \geq)$. That is, iff there exists an element of $lbs(X)$, denoted $\bigwedge X \in lbs(X)$, such that $(\forall y \in lbs(X)) \ \bigwedge X \geq y$. Again, $\bigwedge X$, if it exists, is unique, since it is the *biggest* lower bound of X. For example, in the poset $(\mathcal{P}(3), \subseteq)$ for $X = \{\{0\}, \{2\}\}$ we have $\bigvee X = \{0, 2\}$, and $\bigwedge X = \emptyset$.

Many posets $(A, \leq)$ have sups and/or infs for some choices of subsets $X \subseteq A$, leading to useful notions such as that of a poset with top and/or bottom element, a lattice, a chain-complete poset, or a complete lattice. We can gather several of these notions[3] together in the following definition:

Definition 15 *Given a poset $(A, \leq)$, we say that $(A, \leq)$:*

1. *has a* top *element iff $\bigvee A$ exists. We then use $\top_A$, or just $\top$, to denote $\bigvee A$.*
2. *has a* bottom *element iff $\bigwedge A$ exists. We then use $\bot_A$, or just $\bot$, to denote $\bigwedge A$.*

[3] There are many others. The classic textbook in this area is the excellent [7]; there are also many more recent basic and advanced textbooks. For a detailed study of the different families of subsets of a poset for which one can require to have sups (or dually infs), and the corresponding category-theoretic properties of the complete posets thus obtained see [40].

3. *is a* lattice *iff it has a top and a bottom element,* $\top_A$ *and* $\bot_A$*, and for any two elements* $a, b \in A$*, both* $\bigvee\{a, b\}$ *and* $\bigwedge\{a, b\}$ *exist. We then use the notation* $a \vee b = \bigvee\{a, b\}$*, and* $a \wedge b = \bigwedge\{a, b\}$*.*

4. *is* chain-complete *iff for every subposet* $(C, \leq|_C) \subseteq (A, \leq)$ *such that* $(C, \leq|_C)$ *is a chain, the sup* $\bigvee C$ *exists in* $(A, \leq)$*. The sup* $\bigvee C$ *is usually called the* limit *of the chain* C*.*

5. *is a* complete lattice *iff for any subset* $X \subseteq A$ *both* $\bigvee X$ *and* $\bigwedge X$ *exists. This in particular implies that both* $\top_A = \bigvee A$*, and* $\bot_A = \bigwedge A$ *exist.*

Note that if $(A, \leq)$ is a lattice (resp., complete lattice), then $(A, \geq)$ is also a lattice (resp., complete lattice) with all the operations *dualized*, that is, $\top$ becomes $\bot$ and $\bot$ becomes $\top$, $\bigvee$ becomes $\bigwedge$ and $\bigwedge$ becomes $\bigvee$, and $\vee$ becomes $\wedge$ and $\wedge$ becomes $\vee$.

Note also that in any complete lattice $(A, \leq)$, if $X \subseteq A$, it follows trivially from the definitions of $ubs(X)$ and $lbs(X)$, and of $\bigvee X$ and $\bigwedge X$ that we have the identities:

$$\begin{aligned} \bigvee X &= \bigwedge ubs(X) \\ \bigwedge X &= \bigvee lbs(X) \end{aligned}$$

It also follows trivially (one could fittingly say "vacuously") from the definitions of *ubs* and *lbs* that we have $ubs(\emptyset) = lbs(\emptyset) = A$. Therefore, we have the, at first somewhat confusing, identities: $\bigvee \emptyset = \bigwedge A = \bot$, and $\bigwedge \emptyset = \bigvee A = \top$. However, they should not be *too* confusing, since we have already encountered them in the case of the complete lattice (indeed, complete boolean algebra) $(\mathcal{P}(X), \subseteq)$, where $\bigvee = \bigcup$ and $\bigwedge = \bigcap$. Indeed, we saw in §4.3 that in $(\mathcal{P}(X), \subseteq)$ we have $\bigcap \emptyset = X$, where X is the top element of $(\mathcal{P}(X), \subseteq)$, and of course we have $\bigcup \emptyset = \emptyset$, where $\emptyset$ is the bottom element of $(\mathcal{P}(X), \subseteq)$.

Let us consider some examples for the above concepts. For any set X, the poset $(\mathcal{P}(X), \subseteq)$ is a complete lattice and therefore satisfies all the properties (1)–(5). Given any $\mathcal{U} \subseteq \mathcal{P}(X)$ we of course have $\bigvee \mathcal{U} = \bigcup \mathcal{U}$, and $\bigwedge \mathcal{U} = \bigcap \mathcal{U}$. The lattice operations are just the special case of finite unions and finite intersections, i.e., $A \vee B = A \cup B$, and $A \wedge B = A \cap B$. In particular, the powerset $\mathcal{P}(1) = \mathcal{P}(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\} = 2$ is a complete lattice and therefore a lattice with the inclusion ordering. We call 2 the lattice (in fact more than that, the boolean algebra) of *truth values*, with $0 = \emptyset$ interpreted as *false*, and $1 = \{\emptyset\}$ interpreted as *true*. Note that the lattice operations $\vee$ and $\wedge$ are *exactly* logical disjunction and logical conjunction of truth values in $(\mathcal{P}(1), \subseteq) = (2, \subseteq)$. Therefore, there is no confusion possible between the use of $\vee$ and $\wedge$ for logical operations and for lattice operations, since the second use generalizes the first.

For any set X, the poset $(\mathcal{P}_{fin}(X) \cup \{X\}, \subseteq)$ of finite subsets of X plus X itself is a lattice, with $A \vee B = A \cup B$, and $A \wedge B = A \cap B$, but if X is infinite, then $(\mathcal{P}_{fin}(X) \cup \{X\}, \subseteq)$ is *not* a complete lattice.

The interval $[0, 1]$ in the real line, $[0, 1] = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, is a complete lattice with the usual ordering $\leq$ on real numbers. Given a subset $X \subseteq [0, 1]$ we use the notation $\bigvee X = max(X)$, and $\bigwedge X = min(X)$. $max(X)$ is the smallest real number r such that $r \geq y$ for each $y \in X$, and $min(X)$ is the biggest real number r such that $r \leq y$ for each $y \in X$. The lattice operations $\vee$ and $\wedge$ are also called *max* and *min*, with $max(x, y)$ the biggest of x and y, and $min(x, y)$ the smallest of the two. This lattice is the foundation of *fuzzy logic* [55], which is used in the fuzzy control systems of many electronic devices. The basic idea is that $[0, 1]$ can be viewed as a set of "fuzzy truth values," generalizing the standard truth values in $2 = \{0, 1\}$. Now logical disjunction is *max* and logical conjunction is *min*.

The poset $(\mathbb{R}, \leq)$ with *max* and *min* of two real numbers defined again as maximum and minimum is *almost* a lattice, since the only things it lacks to be a lattice are a top and a bottom element. If we add them, using the standard notation $\top = +\infty$, and $\bot = -\infty$, then $(\mathbb{R} \cup \{+\infty, -\infty\}, \leq)$ becomes a *complete* lattice, with $max(X)$ either the smallest real number bigger than all the elements in X, if X is bounded above, or $+\infty$ otherwise, and with $min(X)$ either the biggest real number smaller than all the elements in X, if X is bounded below, or $-\infty$ otherwise.

The poset $(\mathbb{N}, \leq)$ has a bottom element (namely 0), but has no top element. It is also *almost* a lattice, with $\vee = max$ and $\vee = min$; it just lacks a top element. Also, for any nonempty subset $X \subset \mathbb{N}$, $\bigwedge X = min(X)$ is always defined as the smallest natural number in X, but $\bigvee X = max(X)$ is only defined as the biggest number in X if X is *finite* and nonempty, and as 0 if $X = \emptyset$, but is not defined if X is infinite. However, if we add to $\mathbb{N}$ a top element, denoted as usual $\top = \infty$, then $(\mathbb{N} \cup \{\infty\}, \leq)$ becomes a *complete* lattice, where for any infinite subset $X \subset \mathbb{N}$ we have $max(X) = \infty$.

A very useful example of a chain-complete poset is provided by partial functions. If you did Exercise 31, you already know what a partial function is. If you missed the fun of Exercise 31, here is the definition. A *partial function* from A to B, denoted $f : A \rightharpoonup B$, is a relation $f \subseteq A \times B$ such that for each $a \in A$, $f[\{a\}]$ is always either a singleton set or the empty set. The set $[A \rightharpoonup B]$ of all partial functions from A to B is then:

$$[A \rightharpoonup B] = \{f \in \mathcal{P}(A \times B) \mid (\forall a \in A)(\forall b, b' \in B)(((a, b), (a, b') \in f) \Rightarrow b = b')\}.$$

We then have the obvious inclusions $[A \to B] \subseteq [A \rightharpoonup B] \subseteq \mathcal{P}(A \times B)$. Ordered by subset inclusion, the poset $([A \rightharpoonup B], \subseteq)$ is then a *subposet* of the complete lattice $(\mathcal{P}(A \times B), \subseteq)$. $([A \rightharpoonup B], \subseteq)$ has a bottom element, namely $\emptyset$. But as soon as $A \neq \emptyset$ and B has more than one element, $([A \rightharpoonup B], \subseteq)$ cannot be a lattice, and cannot have a top element. Indeed, the set $[A \to B]$ is *exactly* the set of maximal elements of $([A \rightharpoonup B], \subseteq)$; and under those assumptions on A and B, we can always choose $a \in A$, and $b, b' \in B$ with $b \neq b'$, so that we have partial functions $f = \{(a, b)\}$, and $g = \{(a, b')\}$, but $f \cup g = \{(a, b), (a, b')\}$ is *not* a partial function. Indeed, then $ubs\{f, g\} = \emptyset$, where $ubs\{f, g\}$ denotes the set of upper bounds of $\{f, g\}$ in the poset $([A \rightharpoonup B], \subseteq)$. The important fact, however, is that $([A \rightharpoonup B], \subseteq)$ is *chain-complete*. Indeed, let $C \subseteq [A \rightharpoonup B]$ be a chain. Then its limit is the set $\bigcup C$, which is a partial function, since if we have $(a, b), (a, b') \in \bigcup C$, then we must have $f, g \in C$ with $(a, b) \in f$ and $(a, b') \in g$. But since C is a chain, we have either $f \subseteq g$ or $g \subseteq f$. Assuming, without loss of generality, that $f \subseteq g$, then we also have $(a, b) \in g$, and therefore, $b = b'$, so $\bigcup C$ is indeed a partial function.

Given a function $f : A \longrightarrow A$, an element $a \in A$ is called a *fixpoint* of f iff $f(a) = a$. The following theorem, due to Tarski and Knaster, is choke-full with computer science applications:

Theorem 7 *(Tarski-Knaster). If $(A, \leq)$ is a complete lattice, then any monotonic function $f : (A, \leq) \longrightarrow (A, \leq)$ has a fixpoint. Furthermore, any such f has a smallest possible fixpoint.*

Proof. Consider the set $U = \{x \in A \mid f(x) \leq x\}$, and let $u = \bigwedge U$. Therefore, for any $x \in U$, since f is monotonic and by the definition of U, we have $f(u) \leq f(x) \leq x$, which forces $f(u) \leq u$. Therefore, $u \in U$. But since f is monotonic, $f[U] \subseteq U$, and we have $f(u) \in U$. Hence, $f(u) \geq u$. Therefore, $f(u) = u$, so u is our desired fixpoint. Furthermore, u is the smallest possible fixpoint. Indeed, suppose that v is another fixpoint, so that $f(v) = v$. Then $f(v) \leq v$. Therefore, $v \in U$, and thus, $u \leq v$. □

A very useful, well-known variant of the Tarski-Knaster theorem, also choke-full with even more computer science applications, can be obtained by simultaneously relaxing the requirement of $(A, \leq)$ being a complete lattice to just being chain-complete with a bottom; and strengthening instead the condition on f from being just a monotone function to being chain-continuous. A monotone $f : (A, \leq) \longrightarrow (A, \leq)$ is called *chain-continuous* iff for each chain C in $(A, \leq)$ having a limit $\bigvee C$, we have $f(\bigvee C) = \bigvee f(C)$, that is, f preserves limits of chains.

Theorem 8 *If $(A, \leq)$ is a chain-complete poset with a bottom element $\perp$, then any chain-continuous function $f : (A, \leq) \longrightarrow (A, \leq)$ has a fixpoint. Furthermore, any such f has a smallest possible fixpoint.*

Proof. Obviously, $\perp \leq f(\perp)$. Since f is monotonic, we then have a chain $\{f^n(\perp) \mid n \in \mathbb{N}\}$ of the form

$$\perp \ \leq \ f(\perp) \leq \ \ldots \ \leq f^n(\perp) \ \leq f^{n+1}(\perp) \ \leq \ \ldots$$

Then our desired fixpoint is $\bigvee\{f^n(\perp) \mid n \in \mathbb{N}\}$, because, since f is chain-continuous, we have $f(\bigvee\{f^n(\perp) \mid n \in \mathbb{N}\}) = \bigvee f(\{f^n(\perp) \mid n \in \mathbb{N}\}) = \bigvee\{f^{n+1}(\perp) \mid n \in \mathbb{N}\} = \bigvee\{f^n(\perp) \mid n \in \mathbb{N}\}$. Also, $\bigvee\{f^n(\perp) \mid n \in \mathbb{N}\}$ is the smallest possible fixpoint of f, since if a is any other such fixpoint, then $\perp \leq a$ forces $f^n(\perp) \leq f^n(a) = a$. Therefore, $a \in ubs(\{f^n(\perp) \mid n \in \mathbb{N}\})$, and therefore $\bigvee\{f^n(\perp) \mid n \in \mathbb{N}\} \leq a$, as desired. □

One of the great beauties and mysteries of mathematics is that seemingly abstract and extremely general theorems like the one above give rise to extremely concrete and useful applications; in this case, computer science applications, among others. Theorem 8 is at the heart of the semantics of *recursive function definitions* in a programming language. If we define a function by simple recursion or by primitive recursion, we are always sure that the function so defined is a *total* function, and therefore terminates. But a recursive function definition may never terminate, and therefore its extensional meaning or *semantics* in general is not a total function, but a partial one.

The importance of Theorem 8 for the semantics of programming languages is that it can be used to precisely define the partial function associated to a recursive function definition as a *fixpoint*, thus the name

fixpoint semantics for this method, proposed by the so-called *denotational semantics* approach pioneered by Dana Scott and Christopher Strachey (see, e.g., [47, 48, 44]). Let me illustrate the basic idea with a simple example. Consider the following recursive function definition to compute the factorial function on natural numbers:

$$factorial(n) = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \cdot factorial(p(n)) \textbf{ fi}$$

where p denotes the *predecessor* partial function on the natural numbers, so that $p(s(n)) = n$, and $p(0)$ is undefined. The basic idea is to see such a recursive definition as a *functional*, which is just a fancy word for a function whose arguments are other functions. Specifically, we see the recursive definition of factorial as the (total) function:

$$\delta_{factorial} : [\mathbb{N} \rightharpoonup \mathbb{N}] \longrightarrow [\mathbb{N} \rightharpoonup \mathbb{N}] : f \mapsto \lambda n \in \mathbb{N}. \textbf{ if } n = 0 \textbf{ then } 1 \textbf{ else } n \cdot f(p(n)) \textbf{ fi}.$$

It is not hard to check that $\delta_{factorial}$ is monotonic and chain-continuous in the chain-complete poset of partial functions $([\mathbb{N} \rightharpoonup \mathbb{N}], \subseteq)$. Then, the fixpoint semantics of the factorial function definition is precisely the least fixpoint of $\delta_{factorial}$, that is, the function $\bigvee\{\delta^n_{factorial}(\emptyset) \mid n \in \mathbb{N}\}$, which in this example happens to be total, since the factorial function definition terminates. Let us look in some detail at the increasing chain of partial functions

$$\emptyset \subseteq \delta_{factorial}(\emptyset) \subseteq \ldots \subseteq \delta^n_{factorial}(\emptyset) \subseteq \delta^{n+1}_{factorial}(\emptyset) \subseteq \ldots$$

Just by applying the definition of $\delta_{factorial}$ we can immediately see that this is the sequence of partial functions:

$$\emptyset \subseteq \{(0,1)\} \subseteq \ldots \subseteq \{(0,1),(1,2),\ldots,(n,n!)\} \subseteq \{(0,1),(1,2),\ldots,(n,n!),((n+1),(n+1)!)\} \subseteq \ldots$$

Therefore, if we denote by $!\_$ the factorial function, we have, $\bigvee\{\delta^n_{factorial}(\emptyset) \mid n \in \mathbb{N}\} = !\_$, which is the expected fixpoint semantics of the factorial function definition. The above sequence of partial functions has a strong computational meaning, since it represents the successive approximations of the factorial function obtained by a machine implementation computing deeper and deeper *nested function calls* for *factorial*.

Exercise 65 *Call* $(A, \leq)$ *a* complete sup semilattice *(resp., a* complete inf semilattice*) iff for any subset* $X \subseteq A$, $\bigvee X$ *always exists (resp.,* $\bigwedge X$ *always exists). Prove that any complete* sup *semilattice (resp., any complete* inf *semilattice) is always a complete lattice.*

Exercise 66 *Given a set A, consider the following subsets of* $\mathcal{P}(A \times A)$*: (i) TransRel*(A)*, the set of all transitive relations on A; and (ii) EquivRel*(A)*, the set of all equivalence relations on A (see §9.6 below). Prove that* $(TransRel(A), \subseteq)$ *and* $(EquivRel(A), \subseteq)$ *are both complete lattices. Similarly, given a poset* $(A, <)$*, consider the set* $Sub(A, <)$ *of all its subposets, that is the set* $Sub(A, <) = \{(A', <') \in \mathcal{P}(A) \times \mathcal{P}(A \times A) \mid (A', <') \subseteq (A, <) \wedge poset(A', <')\}$*, where, by definition, the predicate* $poset(A', <')$ *holds iff* $(A', <')$ *is a poset. Prove that* $(Sub(A, <), \subseteq)$ *is a complete lattice. (Hint: in each of the three proofs you can cut your work in half using Exercise 65.)*

Exercise 67 *Let* $(A, \leq_A)$ *and* $(B, \leq_B)$ *be posets, and assume that both have a top element (resp., have a bottom element, resp., are lattices, resp., are chain-complete, resp., are complete lattices). Prove that then* $(A \times B, \leq_{A \times B})$ *has also a top element (resp., has a bottom element, resp., is a lattice, resp., is chain-complete, resp., is a complete lattice).*

9.6 Equivalence Relations and Quotients

Given a set A, the identity relation id_A is what we might call the *absolute equality relation* on A. Yet, life is full of situations where we are not interested in absolute equality but in some kind of *relative equality*. Consider, for example, money. Any two one-dollar bills, which are different in the physical, absolute sense, are nevertheless equal from the relative standpoint of paying with them. Similarly, any two quarter coins are equal for paying purposes. In general, any two bills or coins of the same *denomination* are equal for paying purposes, or, if you wish, *equivalent* (equi-valent: of the same value). Money works as an exchange of wealth precisely because we do *not* care about the individual identities of bills or coins, except up to equivalence.

More precisely, let *Money* be the finite set of dollar bills and coins of different denominations currently in circulation. We can define a binary relation $\equiv \subset Money^2$, where $x \equiv y$ iff x and y are money items of the

same denomination. For example, x can be a dollar bill and y can be a dollar coin. Note that this defines a *partition* of the set *Money* into "buckets," with one bucket for money items of value 1 cent (only cent coins go here), 5 cents (only nickels here), another for those of value of 10 cents (only dimes), another for those of value 25 cents (only quarters), another for value 1 dollar (both dollar bills and dollar coins), another for those of value 5 dollars, and so on.

We can generalize this a little by considering the set $\mathcal{P}(Money)$ of subsets of *Money*. Then we say the two subsets $U, V \subseteq Money$ are *equivalent*, denoted $U \equiv V$ iff their value adds up to the same amount. For example, U can be a set of two dollar bills, a quarter, a dime, and three nickels; and V can be a set with a dollar coin and six quarters. They are *equivalent*, that is, they have the same value, namely, 2 dollars and 50 cents. Again, this *partitions* $\mathcal{P}(Money)$ into "buckets," with one bucket for each concrete amount of money. We now have buckets for 1, 5, 10, and 25 cents, but also buckets for other values from 2 up to 99 cents and beyond. In general we have buckets for any amount of money of the form $n.xy$ with n a natural number, and xy two decimal points, and with $n.xy$ smaller than or equal to the total value of the finite set *Money* of money in circulation. Note that there are two buckets containing each a single subset, namely, the bucket containing the empty set (whose value is 0), and the bucket containing the set *Money*, which is the bucket of biggest value.

Definition 16 *Given a set A, an* equivalence relation *on A is a binary relation $\equiv \subseteq A^2$ such that it is reflexive, symmetric, and transitive.*

Obviously, both $\equiv$ on *Money*, and $\equiv$ on $\mathcal{P}(Money)$, are equivalence relations in this precise sense. Similarly, given $n \in \mathbb{N} - \{0\}$, the relation $x \equiv_n y$ defined by the logical equivalence $x \equiv_n y \;\Leftrightarrow\; |x - y| \in \dot{n}$ is an equivalence relation on $\mathbb{N}$ (namely the relation of having the same remainder when divided by n), which partitions the set $\mathbb{N}$ into n "buckets," namely,

$$\mathbb{N}/n = \{\dot{n}, \dot{n} + 1, \ldots, \dot{n} + (n-1)\}$$

(see also Exercise 6).

Given an equivalence relation $\equiv$ on a set A, and given an element $a \in A$, we call the *equivalence class* of a, denoted $[a]_\equiv$, or just $[a]$ if $\equiv$ is understood, the set

$$[a]_\equiv = \{x \in A \mid x \equiv a\}.$$

The equivalence classes are precisely the "buckets" we have been talking about in the above examples. For example, in *Money* we have one equivalence class for each denomination; and in $\mathbb{N}$ we have one equivalence class for each remainder after division by n. Note that it is trivial to show (exercise) that $a \equiv a'$ iff $[a]_\equiv = [a']_\equiv$. Therefore, equivalence classes are of course pairwise disjoint:

Lemma 7 *Given an equivalence relation $\equiv$ on a set A, the set $A/\!\equiv\; = \{[a]_\equiv \in \mathcal{P}(A) \mid a \in A\}$ is a* partition *of A. Conversely, any partition of A, $\mathcal{U} \subseteq \mathcal{P}(A)$ defines an equivalence relation $\equiv_{\mathcal{U}}$ such that $A/\!\equiv_{\mathcal{U}} = \mathcal{U}$. Furthermore, for any equivalence relation $\equiv$ on A we have the equality $\equiv\; = \;\equiv_{A/\equiv}$.*

Proof. If $A = \varnothing$, then the only equivalence relation is $\varnothing = \varnothing \times \varnothing$, and $\varnothing/\varnothing = \varnothing$, which is a partition of $\bigcup \varnothing = \varnothing$. If $A \neq \varnothing$, obviously $A/\!\equiv\; \neq \varnothing$, and $\varnothing \notin A/\!\equiv$, and we just have to see that any two different equivalence classes $[a]_\equiv$ and $[a']_\equiv$ are disjoint. We reason by contradiction. Suppose $[a]_\equiv \neq [a']_\equiv$ and let $a'' \in [a]_\equiv \cap [a']_\equiv$. Then $a \equiv a''$ and $a'' \equiv a'$. Therefore, since $\equiv$ is transitive, $a \equiv a'$. Hence, $[a]_\equiv = [a']_\equiv$, a contradiction.

Conversely, given a partition of A, $\mathcal{U} \subseteq \mathcal{P}(A)$, we define the equivalence relation $\equiv_{\mathcal{U}}$ by means of the logical equivalence

$$a \equiv_{\mathcal{U}} a' \;\Leftrightarrow\; (\exists U \in \mathcal{U})\; a, a' \in U.$$

This is trivially an equivalence relation such that $A/\!\equiv_{\mathcal{U}} = \mathcal{U}$. It is also trivial to check that $\equiv\; = \;\equiv_{A/\equiv}$. □

The above lemma tells us that equivalence relations on a set A and partitions of such a set are essentially the same thing, and are in bijective correspondence. That is, the assignments $\equiv \;\mapsto\; A/\!\equiv$ and $\mathcal{U} \mapsto \equiv_{\mathcal{U}}$ are inverse to each other. Therefore, partitions and equivalence relations give us two equivalent (no pun intended) viewpoints for looking at the same phenomenon of "relative equality." The equivalence relation

viewpoint emphasizes the intuition of two things being equal in such a relational sense. The partition viewpoint emphasizes the fact that this *classifies* the elements of A into disjoint classes.

Indeed, the mapping $a \mapsto [a]_\equiv$ is precisely the *process of classification*, and is in fact a surjective function

$$q_\equiv : A \twoheadrightarrow A/\equiv : a \mapsto [a]_\equiv$$

called the *quotient map* associated to the equivalence relation $\equiv$. We can generalize this a little, and regard *any* function $f : A \longrightarrow B$ as a process of classifying the elements of A, or, what amounts to the same, as a way of defining a notion of relative equality between the elements of A, namely, a and a' are equal according to f iff $f(a) = f(a')$. That is, any $f : A \longrightarrow B$ classifies the elements of A according to the partition $\{f^{-1}(b) \in \mathcal{P}(A) \mid b \in f[A]\}$, whose associated equivalence relation $\equiv_f$ can be characterized by means of the logical equivalence

$$(\forall a, a' \in A)(a \equiv_f a' \;\Leftrightarrow\; f(a) = f(a')).$$

Note that the equivalence relation $\equiv_f$ is *exactly* our old friend $f; f^{-1}$, which we encountered in Exercise 33. That is, for any function f we have the identity $\equiv_f = f; f^{-1}$. Of course, when f is the function $q_\equiv : A \twoheadrightarrow A/\equiv$, we get $\equiv_{q_\equiv} = \equiv$.

In all the examples we have discussed there is always a function f implicitly used for classification purposes. For the money examples the functions are $den : Money \longrightarrow \mathbb{Q}$, mapping each coin or bill to its denomination as a rational number, and $val : \mathcal{P}(Money) \longrightarrow \mathbb{Q}$, mapping each set U of coins and bills to its total value. Likewise, for the residue classes modulo n, the relevant function is $rem_n : \mathbb{N} \longrightarrow \mathbb{N}$, mapping each number x to its remainder after division by n.

The following lemma has a trivial proof, which is left as an exercise.

Lemma 8 *For any binary relation $R \subseteq A \times A$, the smallest equivalence relation $\overline{R}$ on A such that $R \subseteq \overline{R}$ is precisely the relation $\overline{R} = (R \cup R^{-1})^*$.*

Lemma 9 *For any binary relation $R \subseteq A \times A$ and any function $f : A \longrightarrow B$ such that $(\forall (a, a') \in R)\ f(a) = f(a')$, there is a unique function $\widehat{f} : A/\overline{R} \longrightarrow B$ such that $f = q_{\overline{R}}; \widehat{f}$.*

Proof. Since $q_{\overline{R}}$ is surjective, it is epi (see Exercise 35), and therefore $\widehat{f}$, if it exists, is unique. Let us prove that it does exist. We define $\widehat{f} : A/\overline{R} \longrightarrow B : [a]_{\overline{R}} \mapsto f(a)$. This will be a well-defined function if we prove that $a\overline{R}a' \;\Rightarrow\; f(a) = f(a')$, so that the definition of $\widehat{f}$ does not depend on our *choice* of a representative $a \in [a]_{\overline{R}}$. Note that, by our assumption on f, we have $R \subseteq \equiv_f$. and since $\overline{R}$ is the smallest equivalence relation containing R we also have $\overline{R} \subseteq \equiv_f$. Therefore, $a\overline{R}a' \;\Rightarrow\; a \equiv_f a'$. But $a \equiv_f a' \;\Leftrightarrow\; f(a) = f(a')$. Therefore, $a\overline{R}a' \;\Rightarrow\; f(a) = f(a')$, as desired. □

Corollary 1 *(Factorization Theorem). Any function $f : A \longrightarrow B$ factors as the composition $f = q_{\equiv_f}; \widehat{f}; j^B_{f[A]}$, with $q_{\equiv_f}$ surjective, $\widehat{f}$ bijective, and $j^B_{f[A]}$ an inclusion, where $\widehat{f}$ is the unique function associated by Lemma 9 to f and the relation $\equiv_f$. This factorization is graphically expressed by the commutative diagram:*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, two heads, "q_{\equiv_f}"'] & B \\
A/\equiv_f \arrow[r, "\widehat{f}"] & f[A] \arrow[u, hook, "j^B_{f[A]}"']
\end{tikzcd}$$

Proof. Obviously, by Lemma 9 we have a factorization $f = q_{\equiv_f}; \widehat{f}$. Furthermore, $\widehat{f}$ is *injective*, since

$$\widehat{f}([a]_{\equiv_f}) = \widehat{f}([a']_{\equiv_f}) \Leftrightarrow f(a) = f(a') \Leftrightarrow a \equiv_f a' \Leftrightarrow [a]_{\equiv_f} = [a']_{\equiv_f}.$$

Furthermore, $\widehat{f}$ factors as

$$A/\equiv_f \overset{\widehat{f}}{\twoheadrightarrow} f[A] \overset{j^B_{f[A]}}{\hookrightarrow} B$$

But since $\widehat{f} : A/\equiv_f \longrightarrow B$ is injective, by Exercise 35 (2) and (6), then $\widehat{f} : A/\equiv_f \twoheadrightarrow f[A]$ is also injective, and therefore bijective, as desired. □

Exercise 68 *(Exercise 33 revisited). Prove that for any function $f : A \longrightarrow B$ we have the identity of binary relations $f; f^{-1} = \equiv_f$, and the identity of functions $j^B_{f[A]} = f^{-1}; f$. Therefore, the above factorization diagram can be equivalently rewritten as follows:*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, two heads, "q_{f;f^{-1}}"'] & B \\ A/f;f^{-1} \arrow[r, "\widehat{f}"] & f[A] \arrow[u, hook, "f^{-1};f"'] \end{tikzcd}$$

Exercise 69 *Give a precise and succinct description of* all *the equivalence relations on the set* $3 = \{0, 1, 2\}$*, describing each one of them explicitly. (Hint: describing each such equivalence relation as a set of pairs is certainly precise, but* not *succinct).*

Exercise 70 *Let A and B be any two sets, and let $f, g \in [A{\to}B]$. Prove that the relation $f \approx g$ that holds when the functions f and g agree everywhere except possibly at a finite number of arguments, that is, the relation defined by the equivalence*

$$f \approx g \;\Leftrightarrow\; (\exists X \in \mathcal{P}_{fin}(A))\; f|_{A-X} = g|_{A-X}$$

is an equivalence relation on $[A{\to}B]$.

Exercise 71 *Given a finite set A with n elements, give a numeric expression depending on n that counts the total number of equivalence relations on A, and prove its correctness.*

Exercise 72 *(Extends Exercise 28).*

1. *Given sets A and B, use the* (Sep) *axiom to give explicit definitions by means of set theory formulas of the subsets $[A{\to}B]_{inj}, [A{\to}B]_{surj}, [A{\to}B]_{bij} \subseteq [A{\to}B]$ of, respectively, injective, surjective, and bijective functions from A to B.*
2. *If A and B are finite, with respective number of elements n and m, prove that: (i) $[A{\to}B]_{inj} \neq \emptyset$ iff $n \leq m$, (ii) $[A{\to}B]_{surj} \neq \emptyset$ iff $n \geq m$, and (iii) $[A{\to}B]_{bij} \neq \emptyset$ iff $n = m$.*
3. *Under the respective assumptions $n \leq m$, $n \geq m$, and $n = m$, give explicit numeric expressions, using n and m, that calculate the exact number of functions in, respectively, $[A{\to}B]_{inj}$, $[A{\to}B]_{surj}$, and $[A{\to}B]_{bij}$, and prove their correctness. (Hint: use the Factorization Theorem, Exercise 5, and some elementary combinatorics).*

Given the essential identity *Relation = Directed Graph = Transition System*, a natural question to ask is: how does an equivalence relation look like as a graph?

Exercise 73 *Given a set N of nodes, the* complete graph *on N is the relation N^2. Prove that, given a graph (N, G), G is an equivalence relation iff there exists a partition $\mathcal{U}$ of N such that:*

1. *For each $U \in \mathcal{U}$, $G|_U$ is the complete graph on U.*
2. *$G = \bigcup\{G|_U \in \mathcal{P}(N \times N) \mid U \in \mathcal{U}\}$.*

Likewise, it is natural to ask: how does an equivalence relation look like as a transition system?

Exercise 74 *Call a transition system $\mathcal{A} = (A, \to_{\mathcal{A}})$* reversible *iff $(\forall (a, a') \in \to_{\mathcal{A}})\ (a', a) \in \to^*_{\mathcal{A}}$. Intuitively, in a reversible system we can always "undo" the effect of a transition $a \to_{\mathcal{A}} a'$ by taking further transitions.*

Prove that $\mathcal{A} = (A, \to_{\mathcal{A}})$ is reversible iff $\to^*_{\mathcal{A}}$ is an equivalence relation.

Finally, we can give a graph-theoretic interpretation to the smallest equivalence relation $\overline{G}$ generated by a relation G.

Exercise 75 *Given a graph (N, G), and given a node $n \in N$, the set of nodes $n' \in N$ connected to n (including n itself) can be defined as, $[n]_G = \{n' \in N \mid (n, n') \in (G \cup G^{-1})^*\}$. We call $[n]_G$ the* connected component *of node n. Prove the following (in whichever order you think best, but without using Lemma 8 until you give a proof of it; some choices of order can make proofs of other items in (1)–(3) trivial):*

1. *The set $\{[n]_G \in \mathcal{P}(N) \mid n \in N\}$ of connected components of (N, G) is a partition of N.*
2. *Give a detailed proof of Lemma 8.*
3. *We have a set identity $\{[n]_G \in \mathcal{P}(N) \mid n \in N\} = N/\overline{G}$.*

The following two exercises relate the notion of equivalence relation with two other notions, namely, that of *preorder*, and that of *bisimulation*.

Exercise 76 *(Preorders). A* preorder *(also called a* quasi-order*) is a pair $(A, \preceq)$ with A a set, and $\preceq \in \mathcal{P}(A \times A)$ a reflexive and transitive relation on A. Given preorders $(A, \preceq_A)$, $(B, \preceq_B)$, a* monotonic function *$f : (A, \preceq_A) \longrightarrow (B, \preceq_B)$ is, by definition, a relation homomorphism. Prove that:*

1. *$(A, \preceq)$ is a preorder iff $(\preceq)^* = \preceq$. Furthermore, if $f : (A, R) \longrightarrow (B, G)$ is a relation homomorphism, then $f : (A, R^*) \longrightarrow (B, G^*)$ is a monotonic function between preorders.*
2. *For any preorder $(A, \preceq)$: (i) the relation $\equiv_\preceq$ defined by the equivalence $a \equiv_\preceq a' \;\Leftrightarrow\; (a \preceq a' \wedge a' \preceq a)$ is an equivalence relation; (ii) in the quotient set $A/{\equiv_\preceq}$ the relation $\preceq_\preceq$ defined by the equivalence $[a]_{\equiv_\preceq} \preceq_\preceq [a']_{\equiv_\preceq} \Leftrightarrow a \preceq a'$ is a partial order (in the "less than or equal" sense); and (iii) $q_{\equiv_\preceq} : (A, \preceq) \longrightarrow (A/{\equiv_\preceq}, \preceq_\preceq)$ is monotonic and satisfies the following property: for any poset $(B, \preceq_B)$ and any monotonic $f : (A, \preceq) \longrightarrow (B, \preceq_B)$ there is a unique monotonic $\widehat{f} : (A/{\equiv_\preceq}, \preceq_\preceq) \longrightarrow (B, \preceq_B)$ such that $f = q_{\equiv_\preceq}; \widehat{f}$.*

Exercise 77 *(Minimization of Transition Systems and Automata). Let $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ be a transition system (resp. let $\mathcal{A} = (A, L, \rightarrow_\mathcal{A})$ be a labeled transition system). Prove that the biggest possible bisimulation relation $max.bis : \mathcal{A} \Longrightarrow \mathcal{A}$ (resp. the biggest possible labeled bisimulation relation $max.bis : \mathcal{A} \Longrightarrow \mathcal{A}$) defined in Exercise 58–(4) is an equivalence relation. Let us denote it $\equiv_\mathcal{A}$. Prove also that:*

1. *The set $A/{\equiv_\mathcal{A}}$ has a natural structure as a transition system $\mathcal{A}/{\equiv_\mathcal{A}} = (A/{\equiv_\mathcal{A}}, \rightarrow_{\mathcal{A}/\equiv_\mathcal{A}})$, with $[a] \rightarrow_{\mathcal{A}/\equiv_\mathcal{A}} [a']$ iff $a \rightarrow_\mathcal{A} a'$ (show that this definition of the transition relation does not depend on the choices of representatives a and a' in $[a]$ and $[a']$). $\mathcal{A}/{\equiv_\mathcal{A}}$ is called the* minimal transition system *behaviorally equivalent to $\mathcal{A}$. Similarly, for $\mathcal{A} = (A, L, \rightarrow_\mathcal{A})$ a labeled transition system, the set $A/{\equiv_\mathcal{A}}$ has a natural structure as a labeled transition system $\mathcal{A}/{\equiv_\mathcal{A}} = (A/{\equiv_\mathcal{A}}, L, \rightarrow_{\mathcal{A}/\equiv_\mathcal{A}})$, with $[a] \overset{l}{\rightarrow}_{\mathcal{A}/\equiv_\mathcal{A}} [a']$ iff $a \overset{l}{\rightarrow}_\mathcal{A} a'$ (show that this definition of the transition relation does not depend on the choices of representatives a and a' in $[a]$ and $[a']$). $\mathcal{A}/{\equiv_\mathcal{A}}$ is called the* minimal automaton *behaviorally equivalent to $\mathcal{A}$.*
2. *For any transition system $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ (resp. for any labeled transition system $\mathcal{A} = (A, L, \rightarrow_\mathcal{A})$) the quotient map $q_{\equiv_\mathcal{A}} : A \twoheadrightarrow A/{\equiv_\mathcal{A}}$ is a bisimulation $q_{\equiv_\mathcal{A}} : \mathcal{A} \twoheadrightarrow \mathcal{A}/{\equiv_\mathcal{A}}$ (resp. a labeled bisimulation $q_{\equiv_\mathcal{A}} : \mathcal{A} \twoheadrightarrow \mathcal{A}/{\equiv_\mathcal{A}}$).*
3. *For any total bisimulation relation (resp. total labeled bisimulation relation) $H : \mathcal{A} \Longrightarrow \mathcal{B}$ such that H^{-1} is also total, there is an isomorphism of transition systems (resp. isomorphism of labeled transition systems) $f : \mathcal{A}/{\equiv_\mathcal{A}} \overset{\cong}{\longrightarrow} \mathcal{B}/{\equiv_\mathcal{B}}$; that is, bisimilar systems have the same minimization up to isomorphism.*
4. *For $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ a transition system (resp. $\mathcal{A} = (A, L, \rightarrow_\mathcal{A})$ a labeled transition system), $\mathcal{A}/{\equiv_\mathcal{A}}$ is really minimal, in the sense that it cannot be compressed anymore. That is, the equivalence relation $\equiv_{(\mathcal{A}/\equiv_\mathcal{A})}$ is exactly the identity relation on the set $A/{\equiv_\mathcal{A}}$. In particular, if $\mathcal{A}$ has a finite set of states A, the number of states in $A/{\equiv_\mathcal{A}}$ is the smallest possible for a system bisimilar with $\mathcal{A}$.*

9.7 Constructing $\mathbb{Z}$ and $\mathbb{Q}$

To give some flavor for both how set theory is used to define —that is, to build mathematical models of— all mathematical objects of interest, and how useful the notion of a quotient set under an equivalence relation is, I show how the integers $\mathbb{Z}$ and the rationals $\mathbb{Q}$ can be easily defined as quotient sets.

Let us begin with the set $\mathbb{Z}$ of integers. Any integer can be obtained as the difference between two natural numbers. For example, 7 is $7 - 0$, or, equivalently, $15 - 8$. Likewise, -7 is $0 - 7$, or, equivalently, $8 - 15$. This suggests defining the integers as the quotient set $\mathbb{Z} = \mathbb{N}^2/{\equiv}$, where $\equiv$ is the equivalence relation

$$(x, y) \equiv (x', y') \;\Leftrightarrow\; x + y' = x' + y$$

in particular, the number 7 is represented as the equivalence class $[(7, 0)] = [(15, 8)]$, and the number -7 as the equivalence class $[(0, 7)] = [(8, 15)]$. Note that, given any such equivalence class, we can always choose either a representative $(n, 0)$, which we denote by n, or a representative $(0, n)$, which we denote by $-n$ (of course, we denote $(0, 0)$ by $0 = -0$). We can define addition, subtraction, and multiplication of integers in the obvious way: $[(x, y)] + [(x', y')] = [(x + x', y + y')]$, $[(x, y)] - [(x', y')] = [(x + y', y + x')]$, and $[(x, y)] \cdot [(x', y')] = [((x \cdot x') + (y \cdot y'), (x \cdot y') + (y \cdot x'))]$. One of course has to show that the above definitions of addition, subtraction, and multiplication do not depend on the *choice* of representatives $(x, y) \in [(x, y)]$, but this is a trivial exercise.

Once we have the integers, it is equally easy to define the rationals. Each rational number can be represented as a fraction n/m, with $n \in \mathbb{Z}$ and $m \in \mathbb{Z} - \{0\}$, where $0 = [(0,0)]$. Of course the fractions $1/2$, $2/4$, and $3/6$ are all equivalent. This suggests defining the rational numbers as the quotient set $\mathbb{Q} = (\mathbb{Z} \times (\mathbb{Z} - \{0\}))/\equiv$, where $\equiv$ is the equivalence relation

$$(x, y) \equiv (x', y') \;\Leftrightarrow\; x \cdot y' = x' \cdot y$$

and where we typically use the notation $[(x, y)] = x/y$, and usually pick a representative $(x, y) \in [(x, y)]$ such that x/y is an irreducible fraction with y a natural number, which uniquely represents the equivalence class $[(x, y)]$. Addition, subtraction, multiplication, and division of rational numbers is now defined in the obvious way: $[(x, y)] + [(x', y')] = [((x \cdot y') + (y \cdot x'), y \cdot y')]$, $[(x, y)] - [(x', y')] = [((x \cdot y') - (y \cdot x'), y \cdot y')]$, $[(x, y)] \cdot [(x', y')] = [(x \cdot x', y \cdot y')]$, and $[(x, y)]/[(x', y')] = [(x \cdot y', y \cdot x')]$, where in the last definition we must require that $x' \neq 0$. As before, one has to show that the definition of each operation does not depend on the choice of representatives $(x, y) \in [(x, y)]$, again a trivial exercise.

Chapter 10

Case Study: Fixpoint Semantics of Recursive Functions and Lispy

The notion of *computability*, that is, of what it means for a function to be *effectively computable* was a vague, pretheoretic notion until the 1930s. However, several precise *mathematical definitions* of effective computability were given at that time, including Turing machine computability by Alan Turing ([13], https://www.ideals.illinois.edu/handle/2142/17089115–154), lambda-computability by Alonzo Church ([13], 89–107), and the Herbrand-Gödel-Kleene notion of functions definable by recursive equations ([13], 237–252). All these notions, although based on quite different formalisms, were shown to be *equivalent* notions (and equivalent to other notions by Post and Markov), in the precise sense that they all defined the *same* class of effectively computable functions. Since all kinds of data structures can be encoded as numbers by the technique of Gödel numbering, or, in more practical terms, can be given a binary representations in a present-day digital computer (which, again, can be viewed as numbers in binary notation), these theoretical studies focused on defining the class of computable functions of the form $f : \mathbb{N}^n \longrightarrow \mathbb{N}$, with $n > 0$. Since an effective computational process (e.g., the computation of a Turing machine) may never terminate, such computable functions $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ may not be defined for all inputs. That is, we are naturally led into considering *partial* computable functions, even though, in some cases —for example, when they are defined by primitive recursion (see §7.2), or by well-founded recursion (see §14.3)— we can guarantee that they are total.

Computability theory is of course the basis, both historically and mathematically, of computer science. However, it seems fair to say that, of the three equivalent formalisms of Turing Machines, the lambda calculus, and Herbrand-Gödel-Kleene equational recursive definitions, it is the *most low-level* of the three, namely Turing machines, that has had the strongest influence in computer science practice. Indeed, universal Turing machines were further developed into the von Neumann model of a general-purpose digital computer, on which the design of sequential computers has actually been based. Historically, this favoring of the Turing-von-Neumann model had also enormous consequences for programming languages, since *imperative languages*, which are baroque engineering artifacts to program von Neumann architectures, were the ones developed and are still the prevalent ones in use today. All this obscured from view for many years the possibility of using the two other models, the lambda calculus and Herbrand-Gödel-Kleene equational definitions, as bases for *declarative* programmig languages. To make things worse, further developments in mathematical logic, within the so-called *theory of recursive functions*, tended to use simple but restrictive formalisms, such as Turing machines or μ-recursive functions, which were not well-suited for high-level programming purposes (see, e.g., [27, 12]). As a consequence of all these historical accidents, both in computer science and in mathematics, declarative programming languages did not emerge until the 1960s. The first declarative language, pure LISP, was proposed by John Mac Carthy in 1960 [37], and was subsequently implemented by him and his collaborators at MIT [39]. What Mac Carthy was after was a model of computation that was mathematically simple and elegant *and* could be directly used as a high-level programming language for general-purpose computation, including symbolic computation. Not satisfied with any of the models then in vogue, he proposed a new one based on *recursive function definitions* [38]. A pure LISP program is then just a collection of recursive function definitions based on the primitive functions provided

by the LISP language.

How should one be introduced to computability theory? How should a reader of this book be so introduced? There are of course many possible choices; but there are big differences in the choices one can make. And past historical accidents can be perpetuated by such choices. So one should be wary and ask questions. For example, can we use our chosen model of computation to give a precise *semantics* to a programming language? The choice of Turing machines is quite adequate to give semantics to a low-level *machine language*, but horribly inadequate for high-level languages. The model of recursive function definitions has at least three advantages: (i) it has a simple mathematical definition using *fixpoints*; (ii) such a definition can be direclty used to give semantics to realistic high-level declarative languages such as pure LISP; and (iii) fixpoints can also be used to give semantics to other sequential programming languages, including imperative languages, using the methods of denotational semantics [48, 44]. This is not the only choice: for example, equational logic has also advantages (i)–(iii), does not need fixpoints and is more expressive. But recursive function definitions are quite a good choice at this point, and a good entry point into computability theory. In the context of these notes it has the added advantage of essentially (up to a simple translation) including as a special case the primitive recursive functions that we studied in §7.2 and played with in §8. Furthermore, as part of this case study, the entire theory is implemented in the *Lispy* programming language, so that the reader can play and experiment with such definitions for programming purposes.

10.1 Recursive Function Definitions in a Nutshell

The idea of recursive function definitions is that we have a basic *space*, of data elements (see §5.4) and a set of (possibly partial) basic computable functions defined on it, say $\{h_1, \ldots, h_k\}$, called the *base*. We then define a new computable function, say f, by a recursive definition of the form:

$$f(x_1, \ldots, x_n) = exp(x_1, \ldots, x_n)$$

where the expression $exp(x_1, \ldots, x_n)$ can mention: (i) any of the functions $h_1, \ldots, h_k$; (ii) f itself; and (iii) the conditonal operator *ifelse*, where we assume that we have chosen two different constants *tt* and *ff* in our space (to represent "true" and "false"), and then $ifelse(exp_1, exp_2, exp_2)$ evaluates to the value of exp_2 if exp_1 evaluates to *tt*, and to the value of exp_2 if exp_1 evaluates to *ff*.

We can illustrate this notion by choosing as our space the set $\mathbb{N}$ of natural numbers in Peano notation: $0, s(0), s(s(0)), \ldots, n, s(n), \ldots,$ with base of functions $\{0, s, p, eq\}$, with $0 \in \mathbb{N}$ the zero element, s the successor function, $p = s^{-1}$ the predecessor partial function, and eq the binary equality predicate $eq = \{((n, n), s(0)) \mid n \in \mathbb{N}\} \cup \{((n, m), 0) \mid n, m \in \mathbb{N} \wedge n \neq m\}$, and with *tt* chosen as $s(0)$ and *ff* chosen as 0. Then we can define natural number addition by the recursive definition

$$plus(n, m) = ifelse(eq(m, 0), n, plus(s(n), p(m))).$$

Similarly, we can define natural number substraction by the recursive definition

$$minus(n, m) = ifelse(eq(m, 0), n, minus(p(n), p(m))).$$

Intutively, the way we use these definitions to compute a value of a function f so defined for some arguments is by evaluating *exp* instantiated to the given arguments and then evaluating the "calls" to the basic functions in *exp* according to their graph, and the "calls" to f itself in *exp* by repeatedly applying its recursive definition, always bearing in mind that the first argument to an *ifelse* operator should be evaluated before the other arguments, and then only the argument chosen by the conditional has to be evaluated. Let us see three examples of how addition and substraction are evaluated this way:

$$\begin{aligned}
&plus(s(s(0)), s(s(0))) = ifelse(eq(s(s(0)), 0), s(s(0)), plus(s(s(s(0))), p(s(s(0)))))\\
&= ifelse(0, s(s(0)), plus(s(s(s(0))), p(s(s(0))))) = plus(s(s(s(0))), s(0))\\
&= ifelse(eq(s(0), 0), s(s(s(0))), plus(s(s(s(s(0)))), p(s(0))))\\
&= ifelse(0, s(s(s(0))), plus(s(s(s(s(0)))), p(s(0))))\\
&= plus(s(s(s(s(0)))), 0) = ifelse(eq(0, 0), s(s(s(s(0)))), plus(s(s(s(s(s(0))))), p(0)))\\
&= ifelse(s(0), s(s(s(s(0)))), plus(s(s(s(s(s(0))))), p(0))) = s(s(s(s(0)))).
\end{aligned}$$

$minus(s(s(0)), s(s(0))) = ifelse(eq(s(s(0)), 0), s(s(0)), minus(p(s(s(0))), p(s(s(0)))))$
$= ifelse(0, s(s(0)), minus(p(s(s(0))), p(s(s(0))))) = minus(s(0), s(0))$
$= ifelse(eq(s(0), 0), s(0), minus(p(s(0)), p(s(0)))) = ifelse(0, s(0), minus(p(s(0)), p(s(0))))$
$= minus(p(s(0)), p(s(0))) = minus(0, 0) = ifelse(eq(0, 0), 0, minus(p(0), p(0)))$
$= ifelse(s(0), 0, minus(p(0), p(0))) = 0.$

$minus(s(0), s(s(0))) = ifelse(eq(s(s(0)), 0), s(0), minus(p(s(0)), p(s(s(0)))))$
$= ifelse(0, s(0), minus(p(s(0)), p(s(s(0))))) = minus(0, s(0))$
$= ifelse(eq(s(0), 0), 0, minus(p(0), p(s(0)))) = ifelse(0, s(0), minus(p(0), p(s(0))))$
$= minus(p(0), p(s(0))) = undefined$

Note that *minus* is a partial function. For example, it is undefined in the last evaluation because the argument $p(0)$ is undefined. This is a mild case of partiality, since for cases where *minus* is undefined a machine implementation will stop in finite time with no answer or with an error message. But we can easily define partial recursive functions that loop forever for some inputs. We can even define partial recursive functions of zero arguments, what we might call *partial recursive constants*, that loop forever. Here is one:

$$infinity = s(infinity).$$

But besides being able to define partial computable functions, how general is this formalism to define computable functions? It is Turing complete: any function that a Turing Machine can compute can be programmed (much more easily!) with recursive function definitions. This was shown in detail by McCarthy in [38] by showing that any function definable by μ-recursion (a formalism due to Kleene which is equivant to Turing computability: see, e.g., [27, 12]) is also definable by recursive function definitions from the base $\{0, s, p, eq\}$ on the natural numbers $\mathbb{N}$ (actually the base $\{0, s, eq\}$ is enough, since p can be defined from it) [38]. Shouldn't one then be able to define Ackerman's function? Indeed:

$$A(m, n) = ifelse(eq(m, 0), s(n), ifelse(eq(n, 0), A(p(m), s(0)), A(p(m), A(m, p(n))))).$$

Sometimes it is very convenient to define not a single recursive function in isolation, but several by *mutual recursion*. That is, we give a sequence of recursive definitions

$$f_1(x_1, \ldots, x_{n_1}) = exp_1(x_1, \ldots, x_{n_1}), \quad \ldots \quad , f_k(x_1, \ldots, x_{n_k}) = exp_k(x_1, \ldots, x_{n_k})$$

where the expressions $exp_i(x_1, \ldots, x_{n_i})$ can now mention: (i) any of the functions $h_1, \ldots, h_k$; (ii) any of the f_j, $1 \leq j \leq k$; and (iii) the conditional operator *ifelse*. For example, the *odd* and *even* predicates can be defined by mutual recursion as follows:

$$odd(n) = ifelse(eq(n, 0), 0, even(p(n))) \qquad even(n) = ifelse(eq(n, 0), s(0), odd(p(n))).$$

Mutual recursion is of course the most general case, so that we can view the recursive definition of a single function as a mutual recursion where the number k of functions defined is $k = 1$.

10.2 Fixpoint Semantics of Recursive Function Definitions

This section is the heart of this case study. The question it addresses is: how can we use the set theory we have seen so far to define a *mathematical model* of recursive functions, and thus develop a general theory of computability? This will allow us to give a precise mathematical meaning to the informal description of recursive function definitions that we have just seen. Such mathematical modeling is a nontrivial task, because it has to provide precise answers to nonobvious questions such as the following: How can we do it in general, not for a fixed space and base of functions, but for *any* such space and base? How can we define a *language* characterizing precisely what a recursive function definition *is* (only a *vague* description has been given so far). We certainly need a precise syntactic characterization if we are going to use such definitions as a programming language, because the language is going to need a *parser*. And here comes the killer, semantic question: And how do we *know* that such definitions make any sense, that is, that they actually define *functions* that can be uniquely characterized? Let us get to work, first on syntax.

First of all, how can we choose our syntax for the base of functions on which we are going build our recursive definitions? This must be a set of uninterpreted *function symbols*. How can we model that? Of course with a signature! We have already encountered signatures in Definition 2. But since we are assuming a single type for our space of data, we do not need an order-sorted signature $\Sigma = ((S, <), F)$, just an unsorted one. That is, there is a single sort, say s, and we have function symbols of the form $f : s \ldots s \longrightarrow s$. But since s is unique, why do we have to mention it? We do not have to. Just specifying the number $n \geq 0$ of arguments is enough. Therefore, in the unsorted case we can just define a signature Σ to be a set of *pairs* of the form (f, n), with f a function symbol and n the number of its arguments. How can we then characterize the correct syntactic definitions by mutual recursion of the form

$$f_1(x_1, \ldots, x_{n_1}) = exp_1(x_1, \ldots, x_{n_1}), \quad \ldots \quad , f_k(x_1, \ldots, x_{n_k}) = exp_k(x_1, \ldots, x_{n_k})$$

on our chosen base Σ? The heart of the matter is to preciseley characterized what the allowable expressions exp_i are. What are they? Why of course, terms! But on which signature? Exactly on the unsorted signature $\Sigma_{if}^{\mathcal{F}} = \Sigma \cup \{(f_1, n_1), \ldots, (f_{1k}, n_k)\} \cup \{(ifelse, 3)\}$, where we assume that $\Sigma \cap \{(ifelse, 3)\} = \emptyset$, and that all the f_i are fresh new symbols not used either in Σ or in $\{(ifelse, 3)\}$. Therefore, the exact syntactic characterization is that we must have $exp_i(x_1, \ldots, x_{n_i}) \in T_{\Sigma_{if}^{\mathcal{F}}(\{x_1, \ldots, x_{n_i}\})}$, $1 \leq i \leq k$.

Now that we are done with syntax issues we can focus on the semantics. What does it mean to give a space and a base of partial functions on it, interpreting the base of function symbols Σ? This is just giving a (computable) *partial* Σ*-algebra*.

Definition 17 *Given an unsorted signature* Σ*, a* partial Σ-algebra *is a pair* $\mathcal{A} = (A, \{h_{\mathcal{A}}\}_{(h,n)\in\Sigma})$*, where* A *is a set, and where* $\{f_{\mathcal{A}}\}_{(h,n)\in\Sigma}$ *is a family of partial functions such that for each* $(h, n) \in \Sigma$ *we have* $h_{\mathcal{A}} \in [A^n \rightharpoonup A]$*. A partial* Σ*-algebra* $\mathcal{A} = (A, \{h_{\mathcal{A}}\}_{(h,n)\in\Sigma})$ *is called a (total)* Σ-algebra *iff for each* $(h, n) \in \Sigma$ *we have* $h_{\mathcal{A}} \in [A^n \to A]$*. And a partial* Σ-algebra *is called* computable *iff* A *is a space, and for each* $(h, n) \in \Sigma$ *there is a, posibly nonterminating, algorithm for computing* $h_{\mathcal{A}}$*.*

But isn't this circular reasoning? Aren't we going to define a general notion of computable function? How can we use that very notion of computability in our base? No, there is no circularity. In most theories of computation there are some obviously computable functions (for example the successor function on natural numbers and projection functions from cartesian products $\mathbb{N}^n$) that we take as our basis to *generate* the class of all computable functions. What we are doing here is not to privilege the natural numbers as "the" only space, and allow instead any data type A with two different tt and ff values and with some obviously computable (possibly partial) operations on it to serve as our chosen base. This is clearly useful for programmign language design purposes, since in a real language we may want to use other data types besides natural numbers. A more general answer to the above objection is to say that we really do not need to require that our base $\mathcal{A}$ is a computable partial algebra: any partial algebra will do. What we then get is a notion of *relative computability*: assuming that we somehow have a method for performing the basic operations in $\mathcal{A}$, we then get an effective method to compute functions defined by recursive definitions from the base $\mathcal{A}$. That is, $\mathcal{A}$ works as what is called an *oracle*. However, in what follows I will assume that $\mathcal{A}$ is indeed a computable partial algebra.

Now the real question comes. How do we know that, given a base, a collection of mutually recursive function definitions really define a unique family of partial functions? The answer is remarkably simple. Suppose that we are given a partial Σ-algebra $\mathcal{A} = (A, \{f_{\mathcal{A}}\}_{(f,n)\in\Sigma})$, and that we choose different elements $tt, ff \in A$ as our "truth" values. What are then the functions associated to the mutually recursive definitions

$$f_1(x_1, \ldots, x_{n_1}) = t_1(x_1, \ldots, x_{n_1}), \quad \ldots \quad , f_k(x_1, \ldots, x_{n_k}) = t_k(x_1, \ldots, x_{n_k})$$

The answer was already given when we discussed the fixpoint semantics for the *factorial* function in §9.5. We just need to generalize a little the ideas in that example. The key points of the answer are as follows:

1. Each term t_i defines a chain-continuous functional $t_{i\mathcal{A}} : [A^{n_1} \rightharpoonup A] \times \ldots \times [A^{n_k} \rightharpoonup A] \longrightarrow [A^{n_i} \rightharpoonup A]$ for the obvious subset ordering in each $[A^{n_j} \rightharpoonup A]$.

2. Therefore they all together define a chain-continuous functional

$$(t_{1\mathcal{A}}, \ldots, t_{k\mathcal{A}}) : [A^{n_1} \rightharpoonup A] \times \ldots \times [A^{n_k} \rightharpoonup A] \longrightarrow [A^{n_1} \rightharpoonup A] \times \ldots \times [A^{n_k} \rightharpoonup A].$$

3. The tuple of partial functions $(f_{1\mathcal{A}}, \ldots, f_{k\mathcal{A}})$ defined by the above sequence of mutually recursive definitions is then just the minimal fixpoint:

$$(f_{1\mathcal{A}}, \ldots, f_{k\mathcal{A}}) = \bigcup_{n\in\mathbb{N}} (t_{1\mathcal{A}}, \ldots, t_{k\mathcal{A}})^n(\emptyset, \ldots, \emptyset).$$

4. Therefore, by definition, the $(f_{1\mathcal{A}}, \ldots, f_{k\mathcal{A}})$ are the *smallest possible functions* satisfying the recursive equations, i.e., such that:

$$f_{1\mathcal{A}} = t_{1\mathcal{A}}(f_{1\mathcal{A}}, \ldots, f_{k\mathcal{A}}), \ \ldots \ , f_{k\mathcal{A}} = t_{k\mathcal{A}}(f_{1\mathcal{A}}, \ldots, f_{k\mathcal{A}}).$$

All we need to do is to explain point (1), since (2) follows from the observation that if the functions $f_i : (U, \leq_U) \longrightarrow (V_i, \leq_{V_i})$, $1 \leq i \leq k$ are all chain-continuous, then the function $(f_1, \ldots, f_k) : (U, \leq_U) \longrightarrow (V_1 \times \ldots \times V_k, \leq_{V_1\times\ldots\times V_k})$ is also chain-continuos, and points (3)–(4) follow automatically from Theorem 8. The clarification of point (1) amounts to an inductive definition of the functionals $t_{i\mathcal{A}}$ in terms of the following basic functionals:

- $\_;\_ : [A{\rightharpoonup}B] \times [B{\rightharpoonup}C] \longrightarrow [A{\rightharpoonup}C]$, the usual partial function composition.
- $\langle \_, \ldots, \_ \rangle : [B{\rightharpoonup}A] \times \ldots \times [B{\rightharpoonup}A] \longrightarrow [B{\rightharpoonup}A^n]$, where, by definition, $\langle g_1, \ldots, g_n \rangle = \{(b, (a_1, \ldots, a_n)) \in B \times A^n \mid (b, a_1) \in g_1 \wedge \ldots \wedge (b, a_n) \in g_n\}$.
- **ifelse** $: [B{\rightharpoonup}A] \times [B{\rightharpoonup}A] \times [B{\rightharpoonup}A] \longrightarrow [B{\rightharpoonup}A]$, where, by definition, $\mathbf{ifelse}(g, g', g'') = \{(b, a) \in B \times A \mid ((b, tt) \in g \wedge (b, a) \in g') \vee ((b, ff) \in g \wedge (b, a) \in g'')\}$.

Lemma 10 *Under the usual inclusion ordering,* $([X{\rightharpoonup}Y], \subseteq)$*, the above three functionals are chain-continuos.*

Proof. I prove the result in detail for the first functional, and leave the similar proofs of the other two cases as an exercise. Let C be a chain in $[A{\rightharpoonup}B] \times [B{\rightharpoonup}C]$, and let $(f, g) \in [A{\rightharpoonup}B] \times [B{\rightharpoonup}C]$ be its limit. Since the order $\leq$ on $[A{\rightharpoonup}B] \times [B{\rightharpoonup}C]$ is the product order (see Exercise 61), given by $(f', g') \leq (f, g) \ \Leftrightarrow (f' \subseteq f) \wedge (g' \subseteq g)$, $(f, g) = \bigvee C$ exactly means that $f = \bigcup p_1[C]$, and $g = \bigcup p_2[C]$, where p_1, p_2 are the two projection functions from the product, giving rise to corresponding chains $p_1[C]$ and $p_2[C]$ in $[A{\rightharpoonup}B]$ and $[B{\rightharpoonup}C]$. Chain-continuity then means that $f; g = \bigcup_{(f',g')\in C} f'; g'$. Since for any $(f', g') \in C$ we have $(f' \subseteq f) \wedge (g' \subseteq g)$, we trivially have $f'; g' \subseteq f; g$, and therefore $\bigcup_{(f',g')\in C} f'; g' \subseteq f; g$. But in fact this containment is an equality, since for each $(a, c) \in f; g$ there exists a $b \in B$ with $(a, b) \in f \wedge (b, c) \in g$, and since $f = \bigcup p_1[C]$, and $g = \bigcup p_2[C]$, there exist $f' \in p_1[C]$, $g' \in p_2[C]$, such that $(a, b) \in f' \wedge (b, c) \in g'$, so that $(a, c) \in f'; g'$. Therefore, the functional $\_;\_$ is chain-continuous, as desired. □

We are now ready to define the functionals $t_{i\mathcal{A}} : [A^{n_1}{\rightharpoonup}A] \times \ldots \times [A^{n_k}{\rightharpoonup}A] \longrightarrow [A^{n_i}{\rightharpoonup}A]$ for each $1 \leq i \leq k$. Remember that $t_i = t_i(x_1, \ldots, x_{n_i}) \in T_{\Sigma^{\mathcal{F}}_{if}(\{x_1,\ldots,x_{n_i}\})}$, $1 \leq i \leq k$. Therefore, we can just define a functional $t_{\mathcal{A}} : [A^{n_1}{\rightharpoonup}A] \times \ldots \times [A^{n_k}{\rightharpoonup}A] \longrightarrow [A^{n_i}{\rightharpoonup}A]$ inductively for each $t \in T_{\Sigma^{\mathcal{F}}_{if}(\{x_1,\ldots,x_{n_i}\})}$ as follows:

- for $x_j \in \{x_1, \ldots, x_{n_i}\}$, $x_{j\mathcal{A}} = \lambda(g_1, \ldots, g_{n_i}).\ p_j$, where $p_j : A^{n_i} \longrightarrow A : (a_1, \ldots, a_{n_i}) \mapsto a_j$ is the j-th projection function, and where $(g_1, \ldots, g_{n_i}) \in [A^{n_1}{\rightharpoonup}A] \times \ldots \times [A^{n_k}{\rightharpoonup}A]$.
- for $(b, 0) \in \Sigma$, a constant, $b_{\mathcal{A}} = \lambda(g_1, \ldots, g_{n_i}).\ (\lambda(a_1, \ldots, a_{n_i}) \in A^{n_i}.\ b_{\mathcal{A}}(\emptyset))$, where the second occurence of $b_{\mathcal{A}}$ denotes the partial function $b_{\mathcal{A}} \in [A^0{\rightharpoonup}A]$ specified in the partial algebra $\mathcal{A}$, and we identify $A^0 = \{\emptyset\}$.
- for $t = h(t_1, \ldots, t_m)$, with $(h, m) \in \Sigma$, $m \geq 1$,
  $h(t_1, \ldots, t_m)_{\mathcal{A}} = \lambda(g_1, \ldots, g_{n_i}).\ \langle t_{1\mathcal{A}}(g_1, \ldots, g_{n_i}), \ldots, t_{m\mathcal{A}}(g_1, \ldots, g_{n_i}) \rangle ;\ h_{\mathcal{A}}$, where the partial function $h_{\mathcal{A}} \in [A^m{\rightharpoonup}A]$ is specified in the partial algebra $\mathcal{A}$.
- for $t = f_j(t_1, \ldots, t_{n_j})$, $1 \leq j \leq k$,
  $f_j(t_1, \ldots, t_{n_j})_{\mathcal{A}} = \lambda(g_1, \ldots, g_{n_i}).\ \langle t_{1\mathcal{A}}(g_1, \ldots, g_{n_i}), \ldots, t_{n_j\mathcal{A}}(g_1, \ldots, g_{n_i}) \rangle ;\ g_j$.
- for $t = ifelse(t_1, t_2, t_3)$,
  $ifelse(t_1, t_2, t_3)_{\mathcal{A}} = \lambda(g_1, \ldots, g_{n_i}).\ \mathbf{ifelse}(t_{1\mathcal{A}}(g_1, \ldots, g_{n_i}), t_{2\mathcal{A}}(g_1, \ldots, g_{n_i}), t_{3\mathcal{A}}(g_1, \ldots, g_{n_i}))$

The entire justification of point (1), and therefore of the fixpoint semantics of a sequence of mutual recursive function definitions on a base $\mathcal{A}$, is now reduced to proving the following lemma:

Lemma 11 *For any term $t \in T_{\Sigma^{\mathcal{F}}_{if}(\{x_1,\ldots,x_{n_i}\})}$ the functional $t_{\mathcal{A}} : [A^{n_1} \rightharpoonup A] \times \ldots \times [A^{n_k} \rightharpoonup A] \longrightarrow [A^{n_i} \rightharpoonup A]$ is chain-continuous under the usual set containment order for each function space.*

Proof. (Sketch). The proof uses Lemma 10 and reasons by contradiction, by choosing a term $t \in T_{\Sigma^{\mathcal{F}}_{if}(\{x_1,\ldots,x_{n_i}\})}$ of minimal depth for which $t_{\mathcal{A}}$ is not chain-continuous, where we define the depth of term as follows: $depth(x_j) = depth(b) = 1$, and for each $(q, m) \in \Sigma^{\mathcal{F}}_{if}$ with $m \geq 1$, $depth(q(t_1, \ldots, t_m)) = 1 + max(depth(t_1), \ldots, depth(t_m))$. The remaining details are left as an exercise. □

So, what is a *computable function* on a base $\mathcal{A}$? One that we can obtain by *repeated* sequences of sequences of mutually recursive definitions, where each new sequence of mutally recursive function definitions can *use the names of all the recursive functions already defined in earlier definitions*. The point is that after giving a sequence of mutally recursive function definitions, say,

$$f_1(x_1, \ldots, x_{n_1}) = t_1(x_1, \ldots, x_{n_1}), \quad \ldots \quad , f_k(x_1, \ldots, x_{n_k}) = t_k(x_1, \ldots, x_{n_k})$$

we obtain a *larger base signature*, namely, $\Sigma^{\mathcal{F}} = \Sigma \cup \{(f_1, n_1), \ldots, (f_k, n_k)\}$, and a *larger base* on it extending the base $\mathcal{A}$, namely, the partial $\Sigma^{\mathcal{F}}$-algebra $\mathcal{A}' = (A, \{q_{\mathcal{A}'}\}_{(q,n)\in\Sigma'})$, where if $(q, n) \in \Sigma$ we have $q_{\mathcal{A}'} = q_{\mathcal{A}}$, and for the new $(f_1, n_1), \ldots, (f_{1k}, n_k)$ we have $f_{j\mathcal{A}'} = f_{j\mathcal{A}}$, $1 \leq j \leq k$. Therefore we are always ready for an encore! After each definition we can do it all over again, by adding the just-defined partial functions to our base and then giving a new sequence of mutually recursive definitions that can mention these previously defined functions.

Exercise 78 *Recall the recursive definition of the odd and even predicates in §10.1. Use the formula $(f_{1\mathcal{A}}, \ldots, f_{k\mathcal{A}}) = \bigcup_{n\in\mathbb{N}}(t_{1\mathcal{A}}, \ldots, t_{k\mathcal{A}})^n(\emptyset, \ldots, \emptyset)$ to compute the finite partial functions (as explicit finite sets of pairs of numbers) approximating that are defined by the successive iterations of the fixpoint construction for the odd and even functions up to four iterations. Can you predict how many iterations of the fixpoint construction you would need to be able to compute odd(n) and even(n) for a chosen n?*

Exercise 79 *(No need to broaden the base). Assume a partial Σ-algebra $\mathcal{A}$ as a base, and let*

$$f_1(x_1, \ldots, x_{n_1}) = t_1(x_1, \ldots, x_{n_1}), \quad \ldots \quad , f_k(x_1, \ldots, x_{n_k}) = t_k(x_1, \ldots, x_{n_k})$$

be a sequence of mutually recursive definitions on $\mathcal{A}$, and

$$f_{k+1}(x_1, \ldots, x_{n_{k+1}}) = t_{k+1}(x_1, \ldots, x_{n_{k+1}}), \quad \ldots \quad , f_{k+k'}(x_1, \ldots, x_{n_{k+k'}}) = t_{k+k'}(x_1, \ldots, x_{n_{k+k'}})$$

a second such sequence where the terms $t_{k+1}(x_1, \ldots, x_{n_{k+1}}), \quad \ldots \quad , t_{k+k'}(x_1, \ldots, x_{n_{k+k'}})$, can mention the operations in Σ, the $f_1, \ldots, f_k$, and ifelse. Prove that the fixpoint semantics for the $f_1, \ldots, f_k$ on the base $\mathcal{A}$, and the fixpoint semantics of the $f_{k+1}, \ldots, f_{k+k'}$ on the extended base $\mathcal{A}'$ as defined above, both coincide with the fixpoint semantics of the single combined sequence of mutually recursive definitions

$$f_1(x_1, \ldots, x_{n_1}) = t_1(x_1, \ldots, x_{n_1}), \quad \ldots \quad , f_{k+k'}(x_1, \ldots, x_{n_{k+k'}}) = t_{k+k'}(x_1, \ldots, x_{n_{k+k'}})$$

on the original base $\mathcal{A}$.

10.3 The Lispy Programming Language

Chapter 11

Sets Come in Different Sizes

Obviously, two sets A and B such that $A \cong B$, that is, such that there is a bijection $f : A \xrightarrow{\cong} B$ are essentially the same set, since their elements are placed by f in a one-to-one correspondence. Therefore, the relation $A \cong B$ gives us a precise way of capturing the intuition that A and B have the *same size*.

Definition 18 *(Finite and Infinite Sets). A set A is called* finite *iff there is an $n \in \mathbb{N}$ such that $A \cong n$. A set A is called* infinite *iff it is not finite.*

The relation $A \cong B$ is sometimes called the *equinumerosity* (or *equipotence*) relation. In particular, it is trivial to prove (exercise) that if A is finite, $A \cong B$ iff B is finite and has exactly the same number n of elements as A. That is, any finite set can always be placed in one-to-one correspondence with exactly one $n \in \mathbb{N}$.

How can we capture the notion of a set A having a size *smaller* than (or equal to) that of another set B? Obviously this means that B must contain a subset B' such that $A \cong B'$. But this is equivalent to the existence of an *injective* function $f : A \rightarrowtail B$, since then $A \cong f[A]$, and $f[A] \subseteq B$. Therefore we can introduce the notation $A \preceq B$ to abbreviate the existence of an injective function $f : A \rightarrowtail B$, with the intuitive meaning that the size of A is "smaller than or equal to" that of B. Similarly, the notation $A \prec B$ abbreviates $A \preceq B$ and $A \not\cong B$, that is, the size of A is *strictly smaller* than that of B. For example, for each natural number n we have $n \prec \mathbb{N}$; and for natural numbers n and m, we have $n \prec m$ iff $n < m$.

Note that if A and B are finite sets, and we have $f : A \rightarrowtail B$ with $f[A] \neq B$, then $A \not\cong B$, since A must have strictly fewer elements than B. However, for *infinite* sets we may easily have $f : A \rightarrowtail B$, $f[A] \neq B$, and $A \cong B$. For example, both the successor function $\lambda x \in \mathbb{N}.\ s(x) \in \mathbb{N}$, and the double function $\lambda x \in \mathbb{N}.\ (2 \cdot x) \in \mathbb{N}$ define injective functions $\mathbb{N} \rightarrowtail \mathbb{N}$ whose images do not fill all of $\mathbb{N}$. The image of $\lambda x \in \mathbb{N}.\ s(x) \in \mathbb{N}$ is the set of nonzero natural numbers, and the image of $\lambda x \in \mathbb{N}.\ (2 \cdot x) \in \mathbb{N}$ is the set of even numbers.

Richard Dedekind took this fact of a set being in bijection with one of its proper subsets as the very definition of an infinite set. That is, he proposed the following:

Definition 19 *A set A is called* Dedekind-infinite *iff it has a proper subset $B \subset A$ such that $A \cong B$.*

Trivially, since no finite set can be Dedekind-infinite, any Dedekind-infinite set is infinite in the standard sense of Definition 18. But is every infinite set Dedekind-infinite? The answer to this question will be given in §13.2, Corollary 2.

11.1 Cantor's Theorem

A very reasonable question to ask is: are there different *sizes* for infinite sets? Are some infinite sets intrinsically bigger than others? Intuitively, for example, the "countable infinity" of the natural numbers $\mathbb{N}$ seems to be a smaller size of infinity than the infinity of the "continuum" represented by the real line $\mathbb{R}$ (which we shall see in §15.3 has a bijection $\mathbb{R} \cong \mathcal{P}(\mathbb{N})$). A precise answer to this question, on the affirmative, is given by the following beautiful theorem due to Georg Cantor.

Theorem 9 *(Cantor's Theorem). For any set A we have $A \prec \mathcal{P}(A)$.*

Proof. The function $\{\_\}_A : A \longrightarrow \mathcal{P}(A) : x \mapsto \{x\}$ sending each $x \in A$ to the corresponding singleton set $\{x\} \in \mathcal{P}(A)$ is clearly injective because of extensionality. Therefore we have $A \preceq \mathcal{P}(A)$. We now need to show that $A \ncong \mathcal{P}(A)$, that is, that there is no bijective function $f : A \xrightarrow{\cong} \mathcal{P}(A)$. For this it is enough to show that there is no surjective function $f : A \twoheadrightarrow \mathcal{P}(A)$. Suppose that such a surjective f were to exist. Consider the set $C = \{x \in A \mid x \notin f(x)\}$. Since f is surjective, there must exist an element $a \in A$ such that $f(a) = C$. We then have two possibilities: either $a \in C$, or $a \notin C$. If $a \in C$, then $a \in f(a)$, and therefore, $a \notin C$, a contradiction. If $a \notin C$, then $a \in f(a)$, and therefore, $a \in C$, again a contradiction. Therefore f cannot be surjective. □

Note that Cantor's theorem immediately gives us an infinite ascending chain of strictly bigger infinite sets, namely,

$$\mathbb{N} \prec \mathcal{P}(\mathbb{N}) \prec \mathcal{P}^2(\mathbb{N}) \prec \ldots \prec \mathcal{P}^n(\mathbb{N}) \prec \mathcal{P}^{n+1}(\mathbb{N}) \prec \ldots$$

where, by definition, $\mathcal{P}^{n+1}(\mathbb{N}) = \mathcal{P}(\mathcal{P}^n(\mathbb{N}))$.

11.2 The Schroeder-Bernstein Theorem

Another very simple but important question to ask is the following. Suppose we have two sets A and B such that we have been able to show $A \preceq B$ and $B \preceq A$. Can we then conclude that $A \cong B$? The answer, in the affirmative, is another key theorem of set theory, namely, the Schroeder-Bernstein Theorem. Proofs of this theorem can be a bit messy; I give below a very simple proof due to Peter Johnstone [29].

Theorem 10 *(Schroeder-Bernstein). Given any two sets A and B, if $A \preceq B$ and $B \preceq A$, then $A \cong B$.*

Proof. By hypothesis we have injections $f : A \rightarrowtail B$ and $g : B \rightarrowtail A$. Let us try to place ourselves in the best possible situation to define a bijection between A and B. The best possible situation would be to find a subset $X \subseteq A$ such that $X = A - g[B - f[X]]$. If such a subset were to exist, we could immediately define a bijection between A and B. Indeed, since f is injective, we could place X in bijective correspondence with its image $f[X]$ using f. But then the complement of $f[X]$ in B, that is, $B - f[X]$ is mapped by g to the complement of X in A, namely, $A - X$. Therefore, since g is also injective, we could place $B - f[X]$ in bijective correspondence with $A - X$ using g, and therefore, $A - X$ in bijective correspondence with $B - f[X]$ using g^{-1}. Therefore, the disjoint union of functions:

$$\{(x, f(x)) \in A \times B \mid x \in X\} \cup \{(g(y), y) \in A \times B \mid y \in B - f[X]\}$$

would be our desired bijection, proving $A \cong B$. So all we need to do is to find a subset $X \subseteq A$ such that $X = A - g[B - f[X]]$. This obviously looks like the fixpoint of some function. If we can show that it is the fixpoint of a monotonic function from a complete lattice to itself, we will be done, since Theorem 7 (Tarski-Knaster) guarantees the existence of such a fixpoint. The complete lattice in question is $(\mathcal{P}(A), \subseteq)$, and the monotonic function is just the following composition of monotonic functions:

$$(\mathcal{P}(A), \subseteq) \xrightarrow{f[\_]} (\mathcal{P}(B), \subseteq) \xrightarrow{B-(\_)} (\mathcal{P}(B), \supseteq) \xrightarrow{g[\_]} (\mathcal{P}(A), \supseteq) \xrightarrow{A-(\_)} (\mathcal{P}(A), \subseteq)$$

where one should note the double reversal of the subset orderings due to the second and fourth functions, which perform complementation in, respectively, $\mathcal{P}(B)$ and $\mathcal{P}(A)$. □

Exercise 80 *Prove that for any A, $A \preceq A$. Prove that if $A \preceq B$ and $B \preceq C$ then $A \preceq C$. Prove that if $A \prec B$ and $B \prec C$ then $A \prec C$.*

Chapter 12

I-Indexed Sets

I-indexed sets, are, intuitively, families of sets $\{A_i\}_{i\in I}$ indexed or parameterized by the elements of another set I. For example, if $I = 5$, we could consider the I-indexed set $\{A_i\}_{i\in 4}$, where $A_0 = 13$, $A_1 = 7$, $A_2 = \mathbb{N}$, $A_3 = 7$, and $A_4 = \varnothing$. So, we can think of $\{A_i\}_{i\in I}$ as the listing or sequence of sets

$$13,\ 7,\ \mathbb{N},\ 7,\ \varnothing$$

which is of course *different* from the set $\{\varnothing, 7, 13, \mathbb{N}\}$, both because the elements in the set $\{\varnothing, 7, 13, \mathbb{N}\}$ are unordered, and because our list has repetitions, like 7.

Under closer inspection, an I-indexed set turns out to be nothing new: just a completely different, but very useful, new *perspective*, deserving a notation of its own, on a *surjective function*.

12.1 I-Indexed Sets *are* Surjective Functions

What is a *sequence*? Consider, for example, the sequence of rational numbers $\{1/s(n)\}_{n\in\mathbb{N}}$, which we would display as

$$1,\ \ 1/2,\ \ 1/3,\ \ 1/4,\ \ \ldots\ \ 1/n,\ \ \ldots$$

and which has 0 as its limit. Or maybe the sequence of even numbers $\{2 \cdot n\}_{n\in\mathbb{N}}$, which we can display as

$$0,\ \ 2,\ \ 4,\ \ 6,\ \ \ldots\ \ 2\cdot n,\ \ \ldots$$

In a similar manner, we could consider *finite* sequences like $\{1/s(n)\}_{n\in k}$, or $\{2\cdot n\}_{n\in k}$, with k a natural number, which would just truncate the above infinite sequences to the first k elements in the sequence.

I ask the question "what is a sequence?" on purpose, as an Occam's razor type of question: do we essentially *need* a new concept to get a precise mathematical definition of a sequence, or is it just a *convenient notation* denoting a concept we already *know*? The answer is that, clearly, we do *not* need any more concepts, since a sequence is just a (sometimes) handy notation to describe a *function*. For example, the sequence $\{1/s(n)\}_{n\in\mathbb{N}}$ is just a convenient, alternative notation for the function $1/s(\_) : \mathbb{N} \longrightarrow \mathbb{Q} : n \mapsto 1/s(n)$. Likewise, $\{2 \cdot n\}_{n\in\mathbb{N}}$ is just a convenient notation for the function $2 \cdot \_ : \mathbb{N} \longrightarrow \mathbb{N} : n \mapsto 2 \cdot n$. In a similar way, the corresponding finite sequences would be convenient notation for the functions $1/s(\_) : k \longrightarrow \mathbb{Q} : n \mapsto 1/s(n)$, and $2 \cdot \_ : k \longrightarrow \mathbb{N} : n \mapsto 2 \cdot n$.

There is, however, a slight indeterminacy between the sequence notation and the corresponding function it denotes, because the sequence notation does *not* mention the codomain of such a function. So, we could instead have considered (in the infinite sequence case, for example) the corresponding *surjective* functions $1/s(\_) : \mathbb{N} \twoheadrightarrow \{1/s(n) \in \mathbb{Q} \mid n \in \mathbb{N}\} : n \mapsto 1/s(n)$, and $2 \cdot \_ : \mathbb{N} \twoheadrightarrow \dot{2} : n \mapsto 2 \cdot n$. Note, by the way, that the *subsets* $\{1/s(n) \in \mathbb{Q} \mid n \in \mathbb{N}\} \subset \mathbb{Q}$, and $\dot{2} \subset \mathbb{N}$ are *totally different* from the sequences that generate them, since in such sets we have lost completely all information about the way in which the elements of the set are *enumerated* by the corresponding sequence.

So, a more careful answer to the above question "what is a sequence?" would be that it is a convenient notation for a *surjective* function from either the set $\mathbb{N}$ of natural numbers (infinite sequence), or from a

natural number k (finite sequence), to some other set. This way, the indeterminacy about the codomain of the function is totally eliminated.

But why do we need to restrict ourselves to countable or finite sequences? Why couldn't we consider the uncountable "sequence" $\{x^2\}_{x\in\mathbb{R}}$ as a convenient notation for the function *square* $: \mathbb{R} \twoheadrightarrow \mathbb{R}_{\geq 0} : x \mapsto x^2$? And why do we have to restrict ourselves to numbers? Given *any* set I, why couldn't we consider, for example, the "sequence" $\{(i,i)\}_{i\in I}$ as a convenient notation for the surjective (and injective) function $\delta_I : I \twoheadrightarrow id_I : i \mapsto (i,i)$, mapping each $i \in I$ to the pair (i,i) in the identity function id_I?

Now we have to remember something very important, namely, that in pure set theory *any set element is itself a set*. Therefore, the elements in all these, increasingly more general kinds of "sequences" like $\{1/s(n)\}_{n\in\mathbb{N}}$, $\{x^2\}_{x\in\mathbb{R}}$, and $\{(i,i)\}_{i\in I}$, *are always "sequences" of sets!* In the first sequence the elements are rational numbers, that can be represented as equivalence classes of integers; in the second sequence they are real numbers, again representable as sets (e.g., as "Dedekind cuts"), and in the third sequence they are ordered pairs, which we have seen are a special kind of sets. But this is *always the case*: in pure set theory the elements of *any* set are always other sets. Furthermore, *any* function $f : A \longrightarrow B$ is always a function of the form $f : A \longrightarrow \mathcal{P}(C)$ for some C. Indeed, let $C = \bigcup B$. Then, by the (*Union*) axiom we have $(\forall b \in B)\ b \subseteq \bigcup B$, and therefore, $(\forall b \in B)\ b \in \mathcal{P}(\bigcup B)$, which implies $B \subseteq \mathcal{P}(\bigcup B)$. Therefore, we have the following factorization of f:

$$A \xrightarrow{f} B \overset{J_B^{\mathcal{P}(\bigcup B)}}{\hookrightarrow} \mathcal{P}(\bigcup B)$$

so that f maps each $a \in A$ to a *subset* $f(a) \subseteq \bigcup B$.

We have now arrived at a very useful general notion of "family of sets," extending that of a sequence indexed by numbers, and co-extensive with that of a surjective function $f : I \twoheadrightarrow T$. Instead of using the, too overloaded, word "sequence," given a set I of indices, we will speak of an *I-indexed set*, or an *I-indexed family of sets*. We write the I-indexed set co-extensive with f as $\{f(i)\}_{i\in I}$. The elements $i \in I$ are called *indices*, since they are used to index the different sets in the given family of sets.

Definition 20 *(I-indexed set). A surjective function $f : I \twoheadrightarrow T$ can be equivalently described, in a sequence-like notation, as the I-indexed family of sets $\{f(i)\}_{i\in I}$. In this notation, we call such a surjective f an I-*indexed set.

I-indexed sets are a generalization of ordinary sets, since we can view an ordinary set X as the 1-indexed set associated to the surjective function $\widetilde{X} : 1 \twoheadrightarrow \{X\} : 0 \mapsto X$. The only difference is that in the 1-indexed set case, there is only one set in the family, whereas in general we have a family of sets (and not just a single set) indexed by I. To emphasize that an I-indexed set is a family of sets indexed by I, we will use the suggestive notation $A = \{A_i\}_{i\in I}$, where A is the name of the entire I-indexed set (that is, the name of a surjective function $A : I \twoheadrightarrow A[I]$), and A_i is the set assigned to the index $i \in I$ (making the slight change of notation of writing A_i instead of $A(i)$). The use of A for the name of the entire I-indexed set helps our intuition that it is "just like an ordinary set," except that it is more general.

In which sense more general? How should we *visualize* an I-indexed set? We can think of it, in F. William Lawvere's words [33], as a *continuously varying set*, thus adding dynamics to set theory. In which sense "varying"? Well, we can think of I as a *parameter* along which the set is varying. For example, if $I = \mathbb{N}$, then we can think of $\mathbb{N}$ as *discrete time*, so that in the $\mathbb{N}$-indexed set $A = \{A_n\}_{n\in\mathbb{N}}$, A_n is the "snapshot" of A at time n.

For example, a digital version of the movie *Casablanca*, which we abbreviate to C, can be mathematically modeled as a family of sets (the movie's frames) indexed by natural numbers corresponding to the different "instants" of the projection (each instant happening each 1/30th of a second). Indeed, Casablanca has naturally this intuitive meaning of a time-varying set. Suppose we have a two-dimensional 3000×5000 pixel screen, with 3000×5000 the cartesian product of the sets 3000 and 5000 (therefore, since $3000 \cdot 5000 = 15000000$, this is a 15 Megapixel screen), where each pixel in the screen is shaded by a shade of grey represented by a 16-bit vector. Such vectors are just the elements of the 16-fold cartesian product 2^{16}. Therefore, each frame in our digital version of Casablanca, for example, the frame number 360, denoted C_{360}, is just a function

$$C_{360} : 3000 \times 5000 \longrightarrow 2^{16}$$

which in computer science would be called a "two-dimensional array of 16-bit numbers."

Figure 12.1: C_t for some $t \in 183600$ in the Casablanca movie (from *The Economist, June 14th-20th, pg.89, 2008.*).

Our version of Casablanca lasts 102 minutes, and the movie projects 30 frames per second. Then we have a total of $102 \cdot 60 \cdot 30 = 183600$ frames. Therefore, Casablanca is a 183600-indexed set $C = \{C_t\}_{t \in 183600}$, where each C_t is precisely the specific function from the cartesian product 3000×5000 to the cartesian product 2^{16} corresponding to the t-th frame in the movie. Of course, C is just another notation for a surjective function $C : 183600 \twoheadrightarrow C[183600]$, where $C[183600] = \{C_t \mid t \in 183600\}$ is the set of all frames in the movie.

There is, again, a crucial distinction between the 183600-indexed set $C = \{C_t\}_{t \in 183600}$, and the set $C[183600] = \{C_t \mid t \in 183600\}$, where all information about the *order* in which the frames are arranged is totally lost. Intuitively, and reverting from digital to celluloid to drive the point home, we can think of the set $C[183600] = \{C_t \mid t \in 183600\}$ as a *bag* in which, after cutting with scissors each of the 183600 frames, we have thrown them all together in a jumbled and chaotic way, with no order whatsoever between them. Of course, Casablanca is just an an example. What this example illustrates is a *general method* to *model mathematically* any digital movie as an indexed set. This can of course be quite useful, since any software for digital videos will implicitly or explicitly manipulate such a mathematical model.

The word "continuously" must be taken with a few grains of salt, since for $I = \mathbb{N}$, or $I = 183600$, we might rather talk of a "discretely varying set," (although our eyes are fooled into seeing Casablanca as a continuously varying set of images). But if we take as parameter set $I = \mathbb{R}_{\geq 0}$, then the expression "continuously varying set" fully agrees with our intuition, since in the varying set $A = \{A_t\}_{t \in \mathbb{R}_{\geq 0}}$, A_t is the "snapshot" of A at continuous time t. For example, we can completely describe the time evolution of a billiard ball on a billiards table from the time when it is hit by a player, say at time 0, as a continuously varying set in exactly this sense, namely, as a $\mathbb{R}_{\geq 0}$-indexed set of the form $B = \{B_t\}_{t \in \mathbb{R}_{\geq 0}}$, where for each $t \in \mathbb{R}_{\geq 0}$, B_t is a pair $B_t = (S_t, (u_t, d_t, i_t))$, where $S_t \subset \mathcal{P}(\mathbb{R}^3)$ is a solid sphere, representing the points in space occupied by the ball at time t, and $(u_t, d_t, i_t) \in (\mathbb{R}^3)^3$ are the coordinates at time t of three chosen points on the surface of the ball, namely, the "up," "down," and "impact" points. At time 0 we choose the three points (u_0, d_0, i_0) to be: (i) the point at the top of the ball as it rests on the table, (ii) the opposite point on the surface of the ball under the table, and (iii) the point of impact chosen by the player to hit the ball (which we idealize as a "point," and we can reasonably assume is different from both u_0 and d_0). Then, the points (u_t, d_t, i_t) will be those points on the surface of the solid sphere S_t where the original points (u_0, d_0, i_0) have been carried by the motion of the ball at time t. Note that S_t tells us where the ball is at time t as a solid sphere, but does not tell us anything about the ball's spin. All spin information is captured by the simpler continuously varying set $\{(u_t, d_t, i_t)\}_{t \in \mathbb{R}_{\geq 0}}$.

Exercise 81 *Prove that the continuously varying set $\{(u_t, d_t, i_t)\}_{t\in\mathbb{R}_{\geq 0}}$ completely determines all aspects of the dynamic evolution of the billiard ball over time. That is, at any time t we can always "reconstruct" the solid sphere S_t from the three points (u_t, d_t, i_t).*

Use the bijection $(\mathbb{R}^3)^3 \cong \mathbb{R}^9$, guaranteed by Exercises 45 and 46, to show that we can, equivalently, completely characterize the dynamic evolution of a billiard ball by the indexed set associated to a surjective function $\widetilde{B} : \mathbb{R}_{\geq 0} \twoheadrightarrow C_B$, where $C_B \subseteq \mathbb{R}^9$ is a curve *in the 9-dimensional euclidean space $\mathbb{R}^9$. This curve is parameterized of course by the time $t \in \mathbb{R}_{\geq 0}$, which is the whole idea of a $\mathbb{R}_{\geq 0}$-indexed set. The representation of the ball's dynamics by our original $\mathbb{R}_{\geq 0}$-indexed set B is very intuitive, but our equivalent representation as the $\mathbb{R}_{\geq 0}$-indexed set $\widetilde{B}$ is much simpler. It illustrates a very general idea used to represent the dynamics of a physical systems in a "phase space." That is, we represent the state of a possibly complex system as a point in an n-dimensional euclidean space, so that its dynamic evolution traces a curve parameterized by the time $t \in \mathbb{R}_{\geq 0}$ in such a space, that is, a $\mathbb{R}_{\geq 0}$-indexed set .*

A somewhat different, but also quite intuitive, way of thinking about an I-indexed set is as what is called a *dependent type* in some functional programming languages. For example, the data type of *rational-valued arrays of length n* for any n is not a single data type, but rather a *family* of data types that *depend* on the value of the length parameter $n \in \mathbb{N}$. It is the $\mathbb{N}$-indexed set $Array(\mathbb{Q}) = \{[n\to\mathbb{Q}]\}_{n\in\mathbb{N}}$. Here it is not helpful to think of the parameter set $\mathbb{N}$ as a set of "times." Instead, we think of it as a set of "sizes."

Yet another family of examples comes from algebraic data type specifications, where the index set I is a set of *names* for the types of a data type. For example, I can be the set of names $I = \{Bool, Nat, Int, Rat\}$. Then, an I-indexed set is an actual *family of data types*, that is, an *interpretation* for the type *names*, assigning a concrete *set* of data elements to each type name. For example, a typical I-indexed set $A = \{A_i\}_{i\in I}$ for $I = \{Bool, Nat, Int, Rat\}$ may have $A_{Bool} = 2$, $A_{Nat} = \mathbb{N}$, $A_{Int} = \mathbb{Z}$, and $A_{Rat} = \mathbb{Q}$. But this is not the only possibility: we may wish to interpret the sort *Nat* as naturals modulo 2, and the sort *Int* as 64-bit integers, and then we could have an I-indexed set $B = \{B_i\}_{i\in I}$ with $B_{Bool} = 2$, $B_{Nat} = \mathbb{N}/2$, $B_{Int} = 2^{64}$, and $B_{Rat} = \mathbb{Q}$.

The key idea common to all these intuitions about, and uses for, an I-indexed set —continuously varying set, dependent type, algebraic data type, and so on— is that I is a *parameter set*, so that $A = \{A_i\}_{i\in I}$ is a family of sets which vary along the parameter set I. We can graphically represent such a parametric dependence of $A = \{A_i\}_{i\in I}$ on I by displaying each set A_i as a vertical "fiber" right above its index element $i \in I$, where the "gaps" in the way the sets A_i are spread out above their indices i indicate those cases where

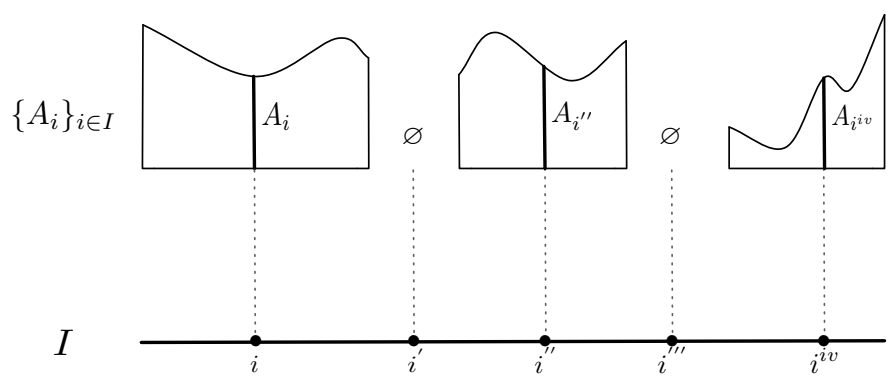


Figure 12.2: Histogram-like display of an I-indexed set $\{A_i\}_{i\in I}$

$A_i = \varnothing$; and where the different sizes of the sets A_i are suggested by their histogram-like display.

It is always instructive to look at corner cases of a definition. Note that in our definition of I-indexed set, the index set I can be *any* set. In particular, we can have the somewhat strange case where $I = \varnothing$. So the question then becomes: how many $\varnothing$-indexed sets are there? The obvious answer is: as many as surjective functions from $\varnothing$ to some other set. But the only such surjective function from $\varnothing$ is the identity function $id_\varnothing : \varnothing \longrightarrow \varnothing$. Therefore, $id_\varnothing$ is the *only* $\varnothing$-indexed set.

Note that the possibility that the sets A_i in an I-indexed set $A = \{A_i\}$ *can* vary does not imply that they *have* to vary. We may happen to have $A_i = A_{i'}$ for all $i, i' \in I$, so that A_i is *constant* and does not change at all as we vary the parameter $i \in I$. For example, we may have an *avant-garde* short movie, called *AVG*, that

Figure 12.3: Based on a painting by Andy Warhol in `http://www.warholprints.com/portfolio/Mao.html`.

lasts 2 minutes and projects during the whole time the single frame $Mao : 3000 \times 5000 \longrightarrow 2^{16}$ shown in Figure 12.1. That is, since $2 \cdot 60 \cdot 30 = 3600$, we have $AVG = \{Mao\}_{t \in 3600}$.

In general, given *any* ordinary set X, we can define the *constant* I-indexed set $X_I = \{X\}_{i \in I}$, that is, the constant family where the set assigned to each $i \in I$ is always the same set X, which is precisely the I-indexed set associated to the constant surjective function $X_I : I \twoheadrightarrow \{X\} : i \mapsto X$. More precisely:

Definition 21 *(Constant I-indexed sets). If $I \neq \varnothing$, then, for any set X, the constant I-indexed set denoted X_I is, by definition, the surjective function $X_I : I \twoheadrightarrow \{X\} : i \mapsto X$. If $I = \varnothing$, then for any set X, the constant $\varnothing$-indexed set denoted $X_\varnothing$ is, by definition, $id_\varnothing$.*

A perceptive reader might have wondered how, in our example of arrays of rational numbers, is the surjective function $Array(\mathbb{Q}) : \mathbb{N} \twoheadrightarrow Array(\mathbb{Q})[\mathbb{N}]$, defining the $\mathbb{N}$-indexed set $Array(\mathbb{Q}) = \{[n \to \mathbb{Q}]\}_{n \in \mathbb{N}}$, precisely defined. This is a nontrivial question, since for such a function to be defined there must exist a set T such that for each $n \in \mathbb{N}$, $[n \to \mathbb{Q}] \in T$. But how do we *know* that such a set exists?

Exercise 82 *Find a set T such that for each $n \in \mathbb{N}$, $[n \to \mathbb{Q}] \in T$. Then define the $\mathbb{N}$-indexed set $Array(\mathbb{Q}) = \{[n \to \mathbb{Q}]\}_{n \in \mathbb{N}}$ explicitly as a surjective function. (Hint: use partial functions).*

For the dependent type of arrays we can indeed find a set T, so that such a dependent type is a surjective function $Array(\mathbb{Q}) : \mathbb{N} \twoheadrightarrow T$. In general, however, we may be able to describe a "family of sets" as a sequence $\{A_i\}_{i \in I}$ without having much of a clue about how to find a set T such that for each $i \in I$ we have $A_i \in T$. How do we know that *in general* such a set T always exists? Note that in Definition 20 an I-indexed set was *defined* to be a surjective function $A : I \twoheadrightarrow A[I]$. Therefore, with that definition the above problem does not explicitly arise, since the family notation $\{A_i\}_{i \in I}$ is just a notational convention for such a function. The unsolved issue, however, is whether given a family $\{A_i\}_{i \in I}$, now taken as the *primary* concept —so that we are not explicitly given a set T such that for each $i \in I$ we have $A_i \in T$— we can always find a surjective function $A : I \twoheadrightarrow A[I]$ such that for each $i \in I$, $A(i) = A_i$. Situations where such a set T is not at all obvious are not hard to come by. Consider, for example, the sequence of sets

$$\varnothing,\ \mathcal{P}(\varnothing),\ \mathcal{P}(\mathcal{P}(\varnothing)),\ \ldots, \mathcal{P}^n(\varnothing),\ \ldots$$

which we could describe as the family of sets (in this alternative sense, where the surjection is not given) $\{\mathcal{P}^n(\varnothing)\}_{n \in \mathbb{N}}$. It is indeed not at all obvious how to find a set T such that for each $n \in \mathbb{N}$ we have $\mathcal{P}^n(\varnothing) \in T$. The answer to the question of whether such a set T always exists will be in the affirmative. However, to

make this answer precise we will need two additional ideas: (i) a notion of *intensional function* that will allow us to precisely charaterize a family $\{A_i\}_{i\in I}$ as a primary notion; and (ii) a new axiom of set theory, called the *replacement axiom*, that will indeed ensure that a correspoding surjective function $A : I \twoheadrightarrow A[I]$ always exists. All this will be developed in detail in §16, which will fully justify the claim that, indeed, I-indexed sets *are* surjective functions. Until §16 we will only use the formulation of Definition 20, where an I-indexed set is explicitly specified by its corresponding surjective function.

Exercise 83 *(I-Indexed Sets as Untyped Functions). The need for requiring I-indexed sets to be surjective functions was due to the indeterminacy of a function's codomain. But this is a side effect of having* typed *the function. Without typing such a problem dissappears. That is, we can define an* untyped function *as a binary relation f (therefore contained in* some *cartesian product, which we need not specify) such that $(\forall x, y, z)\ (x, y), (x, z) \in f \Rightarrow y = z$. Then we can define an I-indexed set as exactly an untyped function f such that $dom(f) = I$, where if, say, $f \subseteq A \times B$, then $dom(f) = \{a \in A \mid (a, b) \in f\}$. Of course, the only possible typing of f as a surjective function is $f : dom(f) \twoheadrightarrow f[dom(f)]$. Check that all we have said so far can be equivalently formulated by taking I-indexed sets to be untyped functions.*

Exercise 84 *(I-Indexed Sets as Surjective Relations). Call a relation $R : X \Longrightarrow Y$* surjective *iff $R[X] = Y$. An alternative way to define an I-indexed set is as a surjective relation $A : I \Longrightarrow B$. We then can use the notation $A = \{A[\{i\}]\}_{i\in I}$ to view A as an S-indexed family of sets. Prove that this, alternative definition of an I-indexed set is* equivalent *to the one in Definition 20 in the following sense:*

1. *By Exercise 47, $A \subseteq I \times B$ with $A[I] = B$ defines a function $\widetilde{A} : I \longrightarrow \mathcal{P}(B) : i \mapsto A[\{i\}]$, and therefore a surjective function $\widetilde{A} : I \twoheadrightarrow \widetilde{A}[I]$ such that, by construction, for each $i \in I$ we have $A(i) = \widetilde{A}(i)$. Therefore, A and $\widetilde{A}$ define the same family of sets.*
2. *Given a surjective function $A : I \twoheadrightarrow T$, let $B = \bigcup T$, and define the relation $\widehat{A} \subseteq I \times B$ by means of the equivalence $(i, b) \in \widehat{A} \Leftrightarrow b \in A(i)$. Obviously, by construction, for each $i \in I$ we have $A(i) = \widehat{A}[\{i\}]$. Therefore, A and $\widehat{A}$ define the same family of sets.*

Prove that $\widehat{A} \subseteq I \times B$ satisfies the equality $\widehat{A}[I] = B$. Prove also the equalities $\widehat{\widetilde{A}} = A$, and $\widetilde{\widehat{A}} = A$, where in the first equality A is a surjective relation, and in the second equality A is a surjective function.

The above proofs then show that the two definitions of an I-indexed set are equivalent. However, the definition of an I-indexed set as a surjective relation is more general, *because it does not depend on the assumption that our set theory is a* pure *set theory, where every element of a set is itself a set. In a version of set theory where some atoms may themselves not be sets, the definition of an I-indexed set as a surjective function would not work, whereas the definition as a surjective relation works perfectly well.*

12.2 Constructing I-Indexed Sets from other I-Indexed Sets

I-indexed sets behave like ordinary sets in many respects. We can perform set-theoretic constructions on them, like union, intersection, disjoint union, cartesian product, and so on, to obtain other I-indexed sets.

The idea is that we can carry out such set-theoretic constructions in a "componentwise" manner: for each index $i \in I$. For example, we can define the union of two I-indexed sets $A = \{A_i\}_{i\in I}$ and $B = \{B_i\}_{i\in I}$ as the I-indexed set $A \cup B = \{A_i \cup B_i\}_{i\in I}$. Note that if X and Y are ordinary sets, and $\widetilde{X}$ and $\widetilde{Y}$ are their corresponding 1-indexed sets, we have the identity: $\widetilde{X} \cup \widetilde{Y} = \widetilde{X \cup Y}$. That is, union of I-indexed sets is an exact *generalization* of union of ordinary sets (which are viewed here as 1-indexed sets). But union is just an example: many other set-theoretic constructions can be generalized to the I-indexed setting in a completely similar manner. Here are some:

Definition 22 *Given I-indexed sets $A = \{A_i\}_{i\in I}$ and $B = \{B_i\}_{i\in I}$, we can define their union, intersection, difference, symmetric difference, cartesian product, disjoint union, function set, and powerset as the following I-indexed sets:*

- $A \cup B = \{A_i \cup B_i\}_{i\in I}$
- $A \cap B = \{A_i \cap B_i\}_{i\in I}$
- $A - B = \{A_i - B_i\}_{i\in I}$
- $A \boxplus B = \{A_i \boxplus B_i\}_{i\in I}$

- $A \times B = \{A_i \times B_i\}_{i \in I}$
- $A \oplus B = \{A_i \oplus B_i\}_{i \in I}$
- $[A \to B] = \{[A_i \to B_i]\}_{i \in I}$
- $\mathcal{P}(A) = \{\mathcal{P}(A_i)\}_{i \in I}$

Similarly, given I-indexed sets $A = \{A_i\}_{i \in I}$ and $B = \{B_i\}_{i \in I}$, we can define their containment relation $A \subseteq B$ by means of the equivalence

$$A \subseteq B \quad \Leftrightarrow \quad (\forall i \in I)\, A_i \subseteq B_i$$

and if this containment relation holds, we call A an *I-indexed subset* of B. How is the empty set generalized to the I-indexed case? It is of course the I-indexed set $\varnothing_I = \{\varnothing\}_{i \in I}$, that is, the constant I-indexed set associated to $\varnothing$. Obviously, the containment relation $\varnothing_I \subseteq A$ holds true for any I-indexed set $A = \{A_i\}_{i \in I}$.

12.3 *I*-Indexed Relations and Functions

How are relations and functions generalized to the I-indexed setting? Given I-indexed sets $A = \{A_i\}_{i \in I}$ and $B = \{B_i\}_{i \in I}$, an *I-indexed relation* R from A to B, denoted $R : A \Longrightarrow B$, is an I-indexed subset $R \subseteq A \times B$. That is, an I-indexed family of relations $R = \{R_i\}_{i \in I}$ such that for each $i \in I$ we have $R_i \subseteq A_i \times B_i$. Similarly, an *$I$-indexed function* from A to B, denoted $f : A \longrightarrow B$, is an I-indexed relation $f = \{f_i\}_{i \in I}$ such that for each $i \in I$, $f_i \in [A_i \to B_i]$. Of course, an I-indexed function $f : A \longrightarrow B$ is called *injective, surjective, or bijective* iff for each $i \in I$ the function f_i is injective, surjective, or bijective. For each I-indexed set $A = \{A_i\}_{i \in I}$, the I-indexed *identity function* id_A is, by definition, $id_A = \{id_{A_i}\}_{i \in I}$.

Also, relation and function composition is defined in the obvious, componentwise way: given I-indexed relations $R : A \Longrightarrow B$ and $G : B \Longrightarrow C$, the I-indexed relation $R; G : A \Longrightarrow C$ is defined componentwise by the equality $R; G = \{R_i; G_i\}_{i \in I}$. Likewise, given I-indexed functions $f : A \longrightarrow B$ and $g : B \longrightarrow C$, the I-indexed function $f; g : A \longrightarrow C$ is defined componentwise by the equality $f; g = \{f_i; g_i\}_{i \in I}$.

The following lemma is a trivial generalization to the I-indexed case of Lemma 4 and is left as an exercise.

Lemma 12 *The following facts hold true for I-indexed relations and functions:*

- *Given I-indexed relations $F : A \Longrightarrow B$, $G : B \Longrightarrow C$, and $H : C \Longrightarrow D$, their composition is* associative*, that is, we have the equality of I-indexed relations $(F; G); H = F; (G; H)$.*
- *Given I-indexed functions $f : A \longrightarrow B$, $g : B \longrightarrow C$, and $h : C \longrightarrow D$, their composition is likewise* associative*, that is, we have the equality of I-indexed functions $(f; g); h = f; (g; h)$.*
- *Given an I-indexed relation $F : A \Longrightarrow B$, we have the equalities $id_A; F = F$, and $F; id_B = F$.*
- *Given an I-indexed function $f : A \longrightarrow B$, we have the equalities $id_A; f = f$, and $f; id_B = f$.*

Let us consider some interesting examples of I-indexed functions. Given $A = \{A_i\}_{i \in I}$ and $B = \{B_i\}_{i \in I}$, consider their I-indexed cartesian product $A \times B$ and disjoint union $A \oplus B$. Then, we have *I-indexed projection functions* $p_1 : A \times B \longrightarrow A$ and $p_2 : A \times B \longrightarrow B$, defined in the obvious, componentwise way, namely, $p_1 = \{p_1 : A_i \times B_i \longrightarrow A_i\}_{i \in I}$ and $p_2 = \{p_2 : A_i \times B_i \longrightarrow B_i\}_{i \in I}$. Similarly, we have the *I-indexed injection functions* into the I-indexed disjoint union, $i_1 : A \longrightarrow A \oplus B$ and $i_2 : B \longrightarrow A \oplus B$, defined by: $i_1 = \{i_1 : A_i \longrightarrow A_i \oplus B_i\}_{i \in I}$ and $i_2 = \{i_2 : B_i \longrightarrow A_i \oplus B_i\}_{i \in I}$.

Similarly as for the case of ordinary functions, we can specify I-indexed functions using lambda expressions. Given I-indexed sets $A = \{A_i\}_{i \in I}$ and $B = \{B_i\}_{s \in S}$, a lambda expression $\lambda i \in I.\lambda x_i \in A_i.\ t(x_i, i) \in B_i$ defines in this way an I-indexed function from $A = \{A_i\}_{i \in I}$ to $B = \{B_i\}_{s \in S}$, provided that we can prove the formula $(\forall i \in I)(\forall x_i \in A_i)\, t(x_i, i) \in B_i$. Indeed, in such a case, for each $i \in I$, the function f_i thus specified is the set of pairs $f_i = \{(a_i, t(a_i, i)) \in A_i \times B_i \mid a_i \in A_i\}$. For example, the I-indexed projection functions $p_1 = \{p_1 : A_i \times B_i \longrightarrow A_i\}_{i \in I}$ and $p_2 = \{p_2 : A_i \times B_i \longrightarrow B_i\}_{i \in I}$, have the obvious lambda expression

specifications $p_1 = \lambda i \in I.\lambda(x_i, y_i) \in A_i \times B_i.\ x_i \in A_i$ and $p_2 = \lambda i \in I.\ \lambda(x_i, y_i) \in A_i \times B_i.\ y_i \in B_i$. Similarly, the *I-indexed injection functions* into the I-indexed disjoint union, $i_1 : A \longrightarrow A \oplus B$ and $i_2 : B \longrightarrow A \oplus B$ have the lambda expression specifications $i_1 = \lambda i \in I.\ \lambda x_i \in A_i.\ (x_i, 0) \in A_i \oplus B_i$, and $i_2 = \lambda i \in I.\ \lambda y_i \in B_i.\ (y_i, 1) \in A_i \oplus B_i$. Likewise, the *I-indexed identity function* $id_A : A \longrightarrow A$ can be specified by the lambda expression $\lambda i \in I.\ \lambda x_i \in A_i.\ x_i \in A_i$.

To give a couple of additional examples, illustrating the use of I-indexed functions as parameterized functions for dependent types, consider the following two $\mathbb{N}$-indexed functions from $Array(\mathbb{Q}) = \{[n{\to}\mathbb{Q}]\}_{n\in\mathbb{N}}$, the dependent type of rational-valued arrays of length n for any n, to, respectively, $\mathcal{P}_{fin}(\mathbb{Q})$, the data type of finite sets of rationals, and $\mathbb{N}$, where the first function maps an array $a = \{(0, x_0), \ldots, (n\text{-}1, x_{n\text{-}1})\}$ to the finite set $\{x_1, \ldots, x_n\}$, and the second function maps an array $a = \{(0, x_0), \ldots, (n\text{-}1, x_{n\text{-}1})\}$ to its length n. Since an ordinary data type can always be viewed as a *constant* dependent type, we can describe these two (parametric in $n \in \mathbb{N}$) functions as $\mathbb{N}$-indexed functions $set : Array(\mathbb{Q}) \longrightarrow \mathcal{P}_{fin}(\mathbb{Q})_{\mathbb{N}}$ and $length : Array(\mathbb{Q}) \longrightarrow \mathbb{N}_{\mathbb{N}}$, defined by the respective lambda expressions: $set = \lambda n \in \mathbb{N}.\lambda a \in [n{\to}\mathbb{Q}].\ a[n] \in \mathcal{P}_{fin}(\mathbb{Q})$, and $length = \lambda n \in \mathbb{N}.\ \lambda a \in [n{\to}\mathbb{Q}].\ n \in \mathbb{N}$.

Exercise 85 *(Generalizes Exercise 39). Given any three I-indexed sets A, B, and C, and given any two I-indexed functions $f : A \longrightarrow C$ and $g : B \longrightarrow C$, we can define the function $[f, g] : A \oplus B \longrightarrow C$ by the defining equation $[f, g] = \{[f_i, g_i]\}_{i\in I}$. Prove that:*

1. $i_1; [f, g] = f$
2. $i_2; [f, g] = g$
3. *(1) and (2) uniquely determine $[f, g]$, that is, any I-indexed function $h : A \oplus B \longrightarrow C$ such that $i_1; h = f$ and $i_2; h = g$ must necessarily satisfy $h = [f, g]$.*

Properties (1)–(3) are compactly expressed by the following commutative diagram of I-indexed functions:

$$\begin{tikzcd}
 & C & \\
A \arrow[ur, "f"] \arrow[r, "i_1"'] & A \oplus B \arrow[u, dotted, "{[f,g]}"] & B \arrow[ul, "g"'] \arrow[l, "i_2"]
\end{tikzcd}$$

Exercise 86 *(Generalizes Exercise 38). Given any three I-indexed sets A, B, and C, and given any two I-indexed functions $f : C \longrightarrow A$ and $g : C \longrightarrow B$, we can define the I-indexed function $(f, g) : C \longrightarrow A \times B$ by means of the defining equation $(f, g) = \{(f_i, g_i)\}_{i\in I}$. Prove that:*

1. $(f, g); p_1 = f$
2. $(f, g); p_2 = g$
3. *(1) and (2) uniquely determine (f, g), that is, any I-indexed function $h : C \longrightarrow A \times B$ such that $h; p_1 = f$ and $h; p_2 = g$ must necessarily satisfy $h = (f, g)$.*

Properties (1)–(3) are compactly expressed by the following commutative diagram of I-indexed functions:

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, dotted, "{(f,g)}"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "p_1"] \arrow[r, "p_2"'] & B
\end{tikzcd}$$

Chapter 13

From I-Indexed Sets to Sets, and the Axiom of Choice

We can gather into a single set data from an I-indexed set in various ways; in particular, the ordinary set $\times A$, which is a generalized cartesian product of all the sets A_i in an I-indexed set $A = \{A_i\}_{i\in I}$, is intimately connected with another key axiom of set theory: the *axiom of choice*.

13.1 Some Constructions Associating a Set to an I-Indexed Set

There are several very useful constructions such that, given an I-indexed set $A = \{A_i\}_{i\in I}$, associate to it an *ordinary set*. Specifically, we will consider four constructions that exactly generalize the four set-theoretic constructions of union, intersection, disjoint union, and cartesian product from constructions on pairs of sets to constructions on arbitrary I-indexed families of sets.

Let us begin with the *union* $\bigcup A$, also denoted $\bigcup_{i\in I} A_i$, of an I-indexed set $A = \{A_i\}_{i\in I}$, which when we view A as a surjective function $A : I \longrightarrow\!\!\!\!\rightarrow A[I]$, is just the union of its codomain set, that is, it is defined by the equation

$$\bigcup A = \bigcup A[I].$$

For example, the union $\bigcup Array(\mathbb{Q})$ of the dependent type $Array(\mathbb{Q}) = \{[n\to\mathbb{Q}]\}_{n\in\mathbb{N}}$ of rational-valued arrays of length n for any n is, of course, the ordinary set $List(\mathbb{Q})$ of *lists* of rational numbers, another very useful data type.

Instead, for $C = \{C_t\}_{t\in 183{,}600}$ the digital version of Casablanca, the set $\bigcup C$ does not seem particularly useful. Since each frame C_t is a function from $3{,}000 \times 5{,}000$ to 2^{16}, and therefore a set of pairs of the form $((i, j), v)$, with $(i, j) \in 3,000 \times 5,000$ and $v \in 2^{16}$, the set $\bigcup C$ is just the set of all such pairs for all frames in the movie. If we find a given pair $((i, j), v)$ in $\bigcup C$, all we know is that for *some* (one or more) frames in the movie, the pixel (i, j) has the shade of gray v.

In a dual way (since union and intersection are dual concepts), if $I \neq \varnothing$ we can define the *intersection* $\bigcap A$, also denoted $\bigcap_{i\in I} A_i$, of an I-indexed set $A = \{A_i\}_{i\in I}$ as the intersection of the codomain set of the function $A : I \longrightarrow\!\!\!\!\rightarrow A[I]$. That is, we define $\bigcap A$ by the equation

$$\bigcap A = \bigcap A[I].$$

For example, the intersection $\bigcap_{n\in\mathbb{N}}[n\to\mathbb{Q}]$ of the dependent type $Array(\mathbb{Q}) = \{[n\to\mathbb{Q}]\}_{n\in\mathbb{N}}$ of rational-valued arrays of length n for any n is, of course, the empty set, since no array can have two different sizes. I strongly conjecture that for $C = \{C_t\}_{t\in 183{,}600}$ the digital version of Casablanca, the intersection $\bigcap C$ is the empty set, since it would be amazing to have a coordinate (i, j) such that its pixel never changes shade in the entire movie! Instead, in the *AVG* movie we have $\bigcap AVG = Mao$. More generally, for any $I \neq \varnothing$, and for any set X, if we consider the constant I-indexed set X_I, we always have $\bigcap X_I = \bigcup X_I = X$.

Yet another very useful construction is the *disjoint* union of an I-indexed set $A = \{A_i\}_{i\in I}$, which is denoted $\bigoplus_{i\in I} A_i$, or just $\bigoplus A$. It is the set defined by the equation

$$\bigoplus_{i\in I} A_i = \{(a, i) \in (\bigcup A) \times I \mid a \in A_i\}.$$

Note that this is an *exact* generalization of the, already defined, binary disjoint union operation $A_0 \oplus A_1$, since for $I = 2$, we can view $A = \{A_n\}_{n\in 2}$ as a 2-indexed set, and then we have the *set identity* $\bigoplus_{n\in 2} A_n = A_0 \oplus A_1$. In this generalization, the two injection functions from A_0 and A_1 to $A_0 \oplus A_1$ are generalized by an I-indexed family of injection functions

$$\{i_i : A_i \longrightarrow \bigoplus A : a_i \mapsto (a_i, i)\}_{i\in I}$$

which we can view as a single I-indexed function $i : A \longrightarrow (\bigoplus A)_I$.

What the $A_0 \oplus A_1$ construction achieved was to first make *disjoint copies* of A_0 and A_1 and then form the union of those copies. This is now generalized to a disjoint union construction for an I-indexed family of sets, so that for each $i, i' \in I$ with $i \neq i'$, the sets $A_i \times \{i\}$ and $A_{i'} \times \{i'\}$ are by construction always disjoint. Therefore, the set $\{A_i \times \{i\} \mid i \in I\}$ is always a *partition* of $\bigoplus_{i\in I} A_i$, and $\bigoplus_{i\in I} A_i$ is just the union of all the sets in such a partition.

For example, the disjoint union $\bigoplus Array(\mathbb{Q}) = \bigoplus_{n\in\mathbb{N}} [n\to\mathbb{Q}]$ of the dependent type of rational-valued arrays of length n for any n, $Array(\mathbb{Q}) = \{[n\to\mathbb{Q}]\}_{n\in\mathbb{N}}$ is the set of pairs (a, n), where a is a rational-valued array and n is its size. Here the disjoint union is not terribly useful, since if $n \neq m$, then $[n\to\mathbb{Q}] \cap [m\to\mathbb{Q}] = \varnothing$. That is, the sets in this family are already pairwise disjoint, so there was really no need to make them pairwise disjoint again. Nevertheless, $\bigoplus Array(\mathbb{Q})$ is a very useful and well-known data type construction in functional programming, where it is called the Σ-*type* (because it is typically denoted $\Sigma Array(\mathbb{Q})$) associated to the dependent type $Array(\mathbb{Q})$.

Instead, for $C = \{C_t\}_{t\in 183{,}600}$ the digital version of Casablanca, the sets in this family are *not* pairwise disjoint: there can be many contiguous frames where relatively few pixels change shade (say, those corresponding to the faces of Humphrey Bogart and Ingrid Bergman), but the pixels corresponding to the furniture and the walls in the room do not change at all. Therefore, in this case the construction $\bigoplus C$ gives us something different from, and more interesting than, the quite useless set of shaded pixels $\bigcup C$. Now the elements of $\bigoplus C$ are pairs $(((i, j), v), t)$, with $((i, j), v)$ the shaded pixel for coordinates (i, j) in the t-th frame. Unlike the case of the set $\bigcup C$, where we had no way to *reconstruct* the movie from $\bigcup C$, we can now reconstruct all of Casablanca (admittedly, in a somewhat painful way) out of the set $\bigoplus C$, since we can recover each frame C_t as the set $C_t = \{((i, j), v) \in \bigcup C \mid (((i, j), v), t) \in \bigoplus C\}$. Of course, this recoverability property holds true not just for Casablanca, but for any disjoint union $\bigoplus_{i\in I} A_i$ for any I-indexed set A, since we can always recover each A_i as the set $A_i = \{a \in \bigcup A \mid (a, i) \in \bigoplus A\}$.

The construction dual to disjoint union is of course that of product. The *product* $\bigtimes_{i\in I} A_i$ of an I-indexed set $A = \{A_i\}_{i\in I}$, which can be abbreviated to just $\bigtimes A$, is defined as the set

$$\bigtimes A = \{a \in [I\to \bigcup A] \mid (\forall i \in I)\ a(i) \in A_i\}.$$

The elements $a \in \bigtimes A$ are sometimes called *choice functions*, since what they do is to *choose* for each index $i \in I$ an element $a(i) \in A_i$ of the corresponding set A_i. The same way that given a surjective function $A : I \twoheadrightarrow A[I]$ we use the sequence notation $A = \{A_i\}_{i\in I}$ for it to emphasize that it is an I-indexed set (making the slight change of notation of writing A_i instead of $A(i)$), given a choice function $a \in \bigtimes A$, we can use the sequence notation $a = \{a_i\}_{i\in I}$ for it, likewise making the slight change of notation of writing a_i instead of $a(i)$. Note that the $\bigtimes A$ construction *generalizes* the cartesian product construction $A_0 \times A_1$, since when $S = 2$, the function

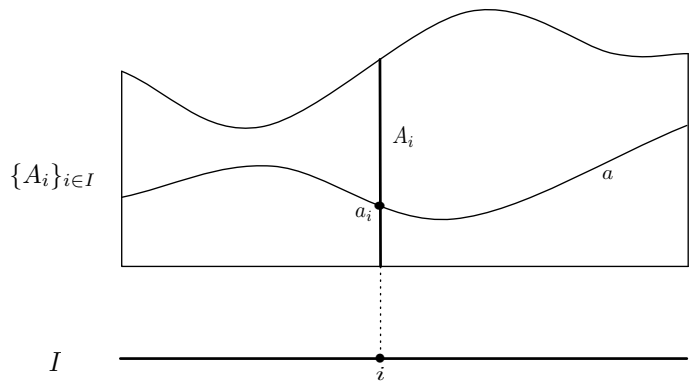
$$\bigtimes_{n\in 2} A_n \longrightarrow A_0 \times A_1 : a \mapsto (a(0), a(1))$$

is well-defined and is a bijection. In this generalization, the two projection functions from $A_0 \times A_1$ to A_0 and A_1 are generalized by an I-indexed family of projection functions

$$\{p_i : \bigtimes A \longrightarrow A_i : a \mapsto a(i)\}_{i\in I}$$

which we can view as a single I-indexed function $p : (\bigtimes A)_I \longrightarrow A$.

A complementary way of thinking of a choice function $a \in \bigtimes A$ is as a *global element* of the I-indexed set $A = \{A_i\}$. That is, each element a_i in the choice function $a = \{a_i\}_{i \in I}$ can be thought of as a *local element* of A, namely, local to the index i and the set A_i. But since $a = \{a_i\}_{i \in I}$ chooses a local element for each local index $i \in I$, it is reasonable to think of a as a *global* element of A. Note that such global elements may not exist at all, that is, we may have $\bigtimes A = \varnothing$. Indeed, this will always happen as soon as there exists an index $i \in I$ such that $A_i = \varnothing$, since then we obviously *cannot* choose a local element for the index i. We can graphically represent choice functions, that is, the elements of $\bigtimes A$ (which we also think of as "global elements" of $A = \{A_i\}_{i \in I}$) as "curves" a lifting each element $i \in I$ to a local element $a_i \in A_i$ in the histogram-like display of $A = \{A_i\}$ shown below



Therefore, the set $\bigtimes A$ is just the set of all such "curves."

Let us consider some examples of global elements. For the dependent type $Array(\mathbb{Q}) = \{[n \to \mathbb{Q}]\}_{n \in \mathbb{N}}$ of rational-valued arrays of length n for any n, its global elements/choice functions are the elements of $\bigtimes Array(\mathbb{Q})$, which in functional programming is called the Π-*type* (a function type which is typically denoted $\Pi Array(\mathbb{Q})$) associated to the dependent type $Array(\mathbb{Q})$. One possible such global element/choice function can be the "infinite sequence of finite sequences[1]" $\{(1, 1/2, 1/3, \ldots, 1/n)\}_{n \in \mathbb{N}}$ of increasingly longer arrays, which we can display as:

$$\varnothing, \quad 1, \quad (1, 1/2), \quad (1, 1/2, 1/3), \quad \ldots (1, 1/2, 1/3, \ldots, 1/n), \ldots$$

and which contains the same information as the infinite sequence of rationals

$$1, \quad 1/2, \quad 1/3, \quad 1/4, \quad \ldots \quad 1/n, \quad \ldots$$

which is here just described as a global element of $Array(\mathbb{Q})$ by listing its successive approximations. A rich source of examples of global elements of $Array(\mathbb{Q})$ which exactly correspond to sequences of rationals comes from the measuring of physical quantities over time. For example, let $temp_{min}$ and $temp_{max}$ be the lowest and highest daily temperatures recorded in the island of Spitsbergen, a matter of some recent concern for polar bears. We can assume that the software of the temperature recording system in the Spitsbergen observatory writes the daily maximum and minimum temperatures in two arrays whose length is incremented each day, so that the two respective sequences of highest and lowest temperatures, and the two respective sequences of arrays of $Array(\mathbb{Q})$ contain the same information. Of course, at each point in time, say day k since the establishment of the Spitsbergen observatory, we do not yet have the entire global element, but only a global element of $Array(\mathbb{Q})\restriction_k = \{[n \to \mathbb{Q}]\}_{n \in k}$.

In general, we can associate an element of $\bigtimes_{n \in \mathbb{N}} [n \to \mathbb{Q}]$ not just to some particular sequences of rationals, but to *any* sequence of rationals. That is, we can define an injective function

$$j : [\mathbb{N} \to \mathbb{Q}] \rightarrowtail \bigtimes_{n \in \mathbb{N}} [n \to \mathbb{Q}] : f \mapsto \{f\restriction_n\}_{n \in \mathbb{N}}$$

which allows us to naturally view each infinite sequence of rationals as an element of $\bigtimes_{n \in \mathbb{N}} [n \to \mathbb{Q}]$. But this injective function from sequences of rationals to global elements of $Array(\mathbb{Q})$ is *not* surjective, since

[1] That is, we display an array $a \in [n \to \mathbb{Q}]$ as the finite sequence $(a(0), a(1), \ldots, a(n\text{-}1))$; then a choice function is an infinite sequence whose elements are finite sequences of increasing length.

other global elements of $Array(\mathbb{Q})$ exist which do not correspond to infinite sequences of rationals at all. For example, we could consider the global element

$$\varnothing, \quad 1, \quad (1, 1/2), \quad (1, 1/4, 1/9), \quad \ldots (1, 1/2^n, 1/3^n, \ldots, 1/n^n), \ldots$$

which does not correspond to any successive approximation of an infinite sequence of rationals. Similarly, the global element

$$\varnothing, \quad 1, \quad (2, 2), \quad (3, 3, 3), \quad \ldots (n, \overset{n}{\ldots}, n), \ldots$$

does not correspond to any successive approximation of an infinite sequence of rationals either.

What about global elements for Casablanca? That is, ordinary elements of $\bigtimes C$? One quite simple family of examples of such elements can be obtained by adopting the extreme myopia of, instead of looking at the whole screen, looking just at a *single pixel*, for example at the pixel with coordinates $(1235, 4736)$. Then the global element

$$\{((1235, 4736), C_t(1235, 4736))\}_{t \in 183600}$$

of Casablanca, which is an ordinary element of $\bigtimes C$, gives us the myopic view of the movie as the sequence of shadings of the pixel with coordinates $(1235, 4736)$ as the movie progresses. We can imagine this as resulting from giving to a movie viewer a very thin tube through which he/she can look at the movie; so thin that he/she can only see one pixel at a time. The above global element, and all the similar global elements that can be obtained by fixing any other coordinate in the cartesian product 3000×5000, corresponds to a viewer that keeps his/her tube *focused always on the same coordinate*. In general, however, we can think of viewers (and of their corresponding global elements) who keep moving their tube around, so that for each particular frame, the single pixel they see the shading of may have a quite different coordinate than those of the pixels they see in prior and subsequent frames.

Exercise 87 *Show that the set $\bigtimes C$ of global elements of Casablanca is the* exact *mathematical description of the different one-pixel "views" that spectators (or a machine working for them) fast enough to focus their one-pixel thin tube at any desired pixel coordinate each 30th of a second would have of that movie.*

The view of choice functions as global elements is also very fruitful in order to relate several set-theoretic constructions, building I-indexed sets out of other I-indexed sets, to familiar notions already encountered, such as those of an I-indexed subset, an I-indexed relation, and an I-indexed function. Recall that in §12.2, given I-indexed sets A and B, we defined other I-indexed sets such as, for example, $\mathcal{P}(A)$, $\mathcal{P}(A \times B)$, and $[A \to B]$. The obvious question to ask about such sets is: what do $\mathcal{P}(A)$, $\mathcal{P}(A \times B)$, and $[A \to B]$ have respectively to do with the notions of an I-indexed subset, an I-indexed relation, and an I-indexed function? And the obvious answer is:

- an I-indexed subset of A is *exactly* an element of $\bigtimes \mathcal{P}(A)$, that is, a global element of $\mathcal{P}(A)$,
- an I-indexed relation from A to B is *exactly* an element of $\bigtimes \mathcal{P}(A \times B)$, that is, a global element of $\mathcal{P}(A \times B)$,
- an I-indexed function from A to B is *exactly* an element of $\bigtimes [A \to B]$, that is, a global element of $[A \to B]$.

Exercise 88 *Given I-indexed sets A and B, show that there are bijections*

$$\bigoplus (A \oplus B) \cong (\bigoplus A) \oplus (\bigoplus B)$$

$$\bigtimes (A \times B) \cong (\bigtimes A) \times (\bigtimes B).$$

Exercise 89 *Given a set X, an I-indexed set A such that $A \subseteq X_I$, and a subset $B \subseteq X$, prove the following set-theoretic equality:*

$$B \cap (\bigcup A) = \bigcup (B_I \cap A).$$

Under the assumption that $I \neq \varnothing$, prove the dual equality

$$B \cup (\bigcap A) = \bigcap (B_I \cup A).$$

Exercise 90 *Prove that for any sets I and B the following two set-theoretic equalities hold:*

1. $\bigoplus B_I = B \times I$
2. $\bigtimes B_I = [I \to B]$

Therefore, we can always think of a cartesian product of two sets as a special case of a disjoint union, and of the function set from I to B as a special case of a product of an I-indexed set. Note that Exercise 45 is a very special case of (2).

Exercise 91 *(Generalizes Exercise 39). Given an I-indexed set A, a set C, and given any I-indexed function $f : A \longrightarrow C_I$, we can define the function $\widehat{f} : \bigoplus A \longrightarrow C : (a_i, i) \mapsto f_i(a_i)$. Prove that:*

1. *for each $i \in I$, $i_i; \widehat{f} = f_i$*
2. *(1) uniquely determines $\widehat{f}$, that is, any function $h : \bigoplus A \longrightarrow C$ such that for each $i \in I$, $i_i; h = f_i$ must necessarily satisfy $h = \widehat{f}$.*

Properties (1)–(2) are graphically expressed by the following commutative diagrams (one for each $i \in I$) of functions:

$$\begin{tikzcd} \bigoplus A \arrow[r, dotted, "\widehat{f}"] & C \\ A_i \arrow[u, "i_i"] \arrow[ur, "f_i"'] & \end{tikzcd}$$

Exercise 92 *(Generalizes Exercise 38). Given an I-indexed set A, a set C, and given any I-indexed function $f : C_I \longrightarrow A$, we can define the function $\widetilde{f} : C \longrightarrow \bigtimes A : c \mapsto \{f_i(c)\}_{i \in I}$. Prove that:*

1. *for each $i \in I$, $\widetilde{f}; p_i = f_i$*
2. *(1) uniquely determines $\widetilde{f}$, that is, any function $h : C \longrightarrow \bigtimes A$ such that for each $i \in I$, $h; p_i = f_i$ must necessarily satisfy $h = \widetilde{f}$.*

Properties (1)–(2) are graphically expressed by the following commutative diagrams (one for each $i \in I$) of functions:

$$\begin{tikzcd} C \arrow[r, dotted, "\widetilde{f}"] \arrow[dr, "f_i"'] & \bigtimes A \arrow[d, "p_i"] \\ & A_i \end{tikzcd}$$

Exercise 93 *(Initial and Final Sets Revisited). Show that for the only $\varnothing$-indexed set $id_\varnothing$ we have:*

1. $\bigoplus id_\varnothing = \varnothing$, *and*
2. $\bigtimes id_\varnothing = 1$.

Show, furthermore, that when we specialize Exercises 91 and 92 to the case when $I = \varnothing$, they exactly *mean, respectively, that $\varnothing$ is the initial set, and that 1 is the final set. This provides a different proof of these two facts, that were already proved in §6.4.*

Exercise 94 *($\bigoplus$ and $\bigtimes$ Are Functorial Constructions) (Generalizes Exercise 41). In a way totally analogous to Exercise 41, we can easily check that $\bigoplus$ and $\bigtimes$ are functorial constructions. Indeed, given an I-indexed function $f : A \longrightarrow B$, we can use Exercise 91 to define the function $\bigoplus f : \bigoplus A \longrightarrow \bigoplus B$ as the unique function associated to the I-indexed function $\{f_i; i_i\}_{i \in I} : A \longrightarrow (\oplus B)_I$. Applying the definition in Exercise 91 it is easy to check that for each $(a_i, i) \in \bigoplus A$ we have $(\bigoplus f)(a_i, i) = (f_i(a_i), i)$.*

The dual definition for $\bigtimes f$ is obtained in the expected way by changing injections by projections and the direction of the arrows. That is, $\bigtimes f : \bigtimes A \longrightarrow \bigtimes B$ is the unique function associated according to Exercise 92 to the I-indexed function $\{p_i; f_i\}_{i \in I} : \bigtimes A \longrightarrow B$. It is then easy to check that $\bigtimes f$ maps each $\{a_i\}_{i \in I} \in \bigtimes A$ to $\{f_i(a_i)\}_{i \in I} \in \bigtimes B$.

Prove that $\bigoplus$ and $\bigtimes$ are functorial constructions, that is, that they also preserve composition and identities. This means that: (i) for any I-indexed functions

$$A \xrightarrow{f} B \xrightarrow{g} C$$

we have $(\bigoplus f); (\bigoplus g) = \bigoplus (f; g)$ (resp., $(\bigtimes f); (\bigtimes g) = \bigtimes (f; g)$); and (ii) for any I-indexed set A we have $\bigoplus id_A = id_{\bigoplus A}$ (resp., $\bigtimes id_A = id_{\bigtimes A}$).

13.2 The Axiom of Choice

The above discussion of choice functions (a.k.a. global elements) has prepared the ground for the axiom of choice (AC), an axiom with far-reaching consequences. Indeed, AC is disputed by some specialists, particularly those with a constructivist bend of mind, who prefer to develop as much of set theory as possible without assuming it. This is the reason for the notation ZFC, which abbreviates $ZF + AC$, so that ZF abbreviates the axioms of Zermelo-Fraenkel set theory *without AC*. The statement of AC is very simple:

> *Any I-indexed set $A = \{A_i\}_{i \in I}$ such that for each $i \in I$ the set A_i is nonempty has a choice function.*

Its precise formalization as a set theory formula is just as simple:

$$(AC) \qquad (\forall I)(\forall A : I \twoheadrightarrow A[I])(((\forall i \in I)\ A(i) \neq \varnothing) \ \Rightarrow\ \bigtimes A \neq \varnothing).$$

At first sight, AC seems "obvious." Why couldn't we choose an element a_i for each set A_i, if all such sets A_i are nonempty? The obvious-looking AC axiom becomes a little less obvious when we reflect on what our notion of a *function* is. If we think of a function $\lambda x \in A.\ t(x) \in B$ as a precise *rule* to assign a given output value $t(x)$ to each input value x, then the issue becomes whether, given an *arbitrary* I-indexed set $A = \{A_i\}_{i \in I}$ with all their A_i nonempty, we can *always* come up with a precise and principled way (a rule in some sense) of choosing for each $i \in I$ an element $a_i \in A_i$.

The difficulty of always being able to come up with such a rule can be illustrated by a thought experiment due to Bertrand Russell, which he presented in one of his books on the philosophy of mathematics [46]. Russell was 98 years old when he died in 1970. Had he lived for a few more years, he might perhaps have considered re-issuing his *Introduction to Mathematical Philosophy* with a new version of his thought experiment, which originally involved a millionaire having infinitely many pairs of boots and socks. The new version, besides illustrating the issues involved in the axiom of choice, could have also served as a morality tale, providing further illustration after the Vietnam war (Russell and Jean-Paul Sartre organized a Vietnam War Crimes Tribunal) of what Russell felt were disastrous consequences of American imperialism.

The revised tale might perhaps have gone as follows:

> *Once upon a time, in an amazing world in which, besides finite things, all kinds of infinite things existed, there lived a woman called Rimmelda. She was an infinitely vain former beauty queen, married to an infinitely rich dictator of an infinitely poor banana republic, controlled from the shadows by an infinitely powerful CIA. Rimmelda never wore the same dress twice: she had an infinite wardrobe, with dresses from the most famous fashion designers, and with an infinite number of pairs of high-heel shoes. She also had an infinite number of pairs of silk stockings in her wardrobe.*
>
> *In planning her future appearance at an infinite number of state banquets, Rimmelda wanted to set in place a clear and unambiguous method by which, when helping her to dress, her maids would hand her first one shoe and a corresponding stocking, and then a second matching shoe and its corresponding matching stocking. With so many shoes and stockings around, this was badly needed, since there had been great confusion among the maids, causing embarrassing mismatches in Rimmelda's attire at some state banquets. Trying to solve this problem, Rimmelda run into serious difficulties, which she vaguely suspected could be mathematical in nature. A mathematics professor from the most important university in the country was summoned to the dictator's palace to solve the problem. This professor was a constructivist who had had a strict personal and mathematical upbringing. He always told the truth, and he only accepted as valid functions those assignments from arguments to values that could be explicitly defined by clear and unambiguous rules. He told Rimmelda that he could provide a clear rule for the shoes: the first shoe chosen should always be the left shoe in the pair; but that no such clear and unambiguous rule could be given for the stockings, since the two stockings in each pair look exactly the same. Rimmelda went into a tantrum, and the professor was accused of secretly plotting a leftist conspiracy and was sent into prison.*

In the language of I-indexed sets we can explain the professor's inability to solve Rimmelda's problem as follows. The infinite collection of pairs of shoes can be described as an $\mathbb{N}$-indexed set *Shoes* =

$\{Shoes_n\}_{n\in\mathbb{N}}$, where $Shoes_n$ is the n-th pair of shoes. Since each set $Shoes_n$ has exactly two elements (the left shoe and the right shoe in the pair), all sets $Shoes_n$ are nonempty. And there is indeed no problem in coming up with a well-principled choice function in $\bigtimes Shoes$: we can, for example, follow the rule of always choosing the *left* shoe in each pair. The infinite collection of pairs of stockings can also be described as an $\mathbb{N}$-indexed set $Stockings = \{Stockings_n\}_{n\in\mathbb{N}}$, with $Stockings_n$ the n-th pair of stockings. Again, since each set $Stockings_n$ has exactly two elements, all sets $Stockings_n$ are nonempty. However, since the two stockings in each pair are *indistinguishable*, there seems to be no clear way of coming up with an unambiguous method or rule for choosing one of the two stockings in each pair. That is, there seems to be no clear way of unambiguously specifying a choice function in $\bigtimes Stockings$.

The point of this story is that AC is in some sense *non-constructive*, since it postulates the existence of a choice function in the product $\bigtimes A$ of any I-indexed set A whose component sets are all nonempty, but provides no explicit description of how such a choice function can be defined. The difficulty does *not* come from being able to pick one stocking in a single pair of stockings: we can always do that. As shown in Exercise 96, there is no difficulty either in choosing an element in $\bigtimes_{i\in I} A_i$ when each of the A_i is nonempty and the index set I is *finite*. The problem comes when I is *infinite*, since then an unambiguous rule or method for the choice should be a *parametric* rule, that can be applied in a *uniform* way (like the rule for picking the left shoe) to an infinite number of instance cases.

The axiom of choice can be further illuminated by another, very intuitive set-theoretic property that is in fact equivalent to it. Recall that, right after Exercise 34, we asked the question:

> *Is every surjective function a right inverse?*

but postponed answering such a question until now. The right answer is that this *depends* on whether we adopt AC as an axiom in our set theory or not. Because, in fact, AC is *equivalent* to the property of every surjective function being a right inverse, as the following theorem shows.

Theorem 11 AC *is equivalent to every surjective function being a right inverse.*

Proof. To prove the ($\Rightarrow$) part, just notice that $f : A \twoheadrightarrow B$ is a surjective function iff the B-indexed set $\{f^{-1}(b)\}_{b\in B}$ is such that for each $b \in B$ the set $f^{-1}[\{b\}]$ is nonempty. By AC, the set $\bigtimes_{b\in B} f^{-1}[\{b\}]$ is then nonempty. But what is $\bigtimes_{b\in B} f^{-1}[\{b\}]$? First notice that $\bigcup\{f^{-1}[\{b\}]\}_{b\in B} = A$. Therefore, $\bigtimes_{b\in B} f^{-1}[\{b\}] \subseteq [B{\to}A]$ is just the set of all functions $g : B \longrightarrow A$ such that for each $b \in B$, $g(b) \in f^{-1}[\{b\}]$. That is, $\bigtimes_{b\in B} f^{-1}[\{b\}]$ is precisely the set of all functions that are *left inverses* to f. Since $\bigtimes_{b\in B} f^{-1}[\{b\}]$ is nonempty, f has a left inverse, and is therefore a right inverse, as desired.

To prove the ($\Leftarrow$) part, let $A = \{A_i\}_{i\in I}$ be an I-indexed set such that for each $i \in I$ we have $A_i \neq \varnothing$. This also implies that for each $i \in I$ the set $A_i \times \{i\} \neq \varnothing$. Consider now $\bigoplus A = \bigcup\{A_i \times \{i\} \mid i \in I\}$, and define the projection function $p_2 : \bigoplus A \longrightarrow I : (a_i, i) \mapsto i$. Since for each $i \in I$ we have $A_i \times \{i\} \neq \varnothing$, p_2 is obviously surjective, and therefore, by our assumption, a right inverse. Let $g : I \longrightarrow \bigoplus A$ be such that $g; p_2 = id_I$. Consider now the projection function $p_1 : \bigoplus A \longrightarrow \bigcup A : (a_i, i) \mapsto a_i$. I claim that $g; p_1$ is a choice function. Indeed, for each $i \in I$ we have $g(i) \in A_i \times \{i\}$, and therefore, $p_1(g(i)) \in A_i$. Therefore, $\bigtimes A \neq \varnothing$, and as a consequence AC holds. □

Note that Theorem 11 gives us an alternative way, given sets A and B, of proving $A \preceq B$. The standard way is of course to exhibit an injective function $f : A \rightarrowtail B$. But we now have the alternative possibility of proving $A \preceq B$ by exhibiting a *surjective* function $g : B \twoheadrightarrow A$, since by Theorem 11 this ensures the existence of a left inverse (and therefore injective, see Exercise 34) function $f : A \rightarrowtail B$ such that $f; g = id_A$.

We can gain additional insights about AC by considering an older, equivalent formulation of it, which we shall call AC':

> (AC') *For every set X there exists a function $c : \mathcal{P}(X) - \{\varnothing\} \longrightarrow X$, from the set of its nonempty subsets to X, such that for each $A \in \mathcal{P}(X) - \{\varnothing\}$ we have $c(A) \in A$.*

The term *choice function* was originally used, in a more restricted sense, for a function c with such a property, since it chooses for each nonempty subset of X an element in that subset.

AC and AC' are in fact equivalent, as shown by the following:

Theorem 12 $AC \Leftrightarrow AC'$.

Proof. For the ($\Rightarrow$) direction, just observe that the identity function $id_{\mathcal{P}(X)-\{\varnothing\}} : \mathcal{P}(X) - \{\varnothing\} \longrightarrow \mathcal{P}(X) - \{\varnothing\}$ is bijective and in particular surjective. Therefore, $id_{\mathcal{P}(X)-\{\varnothing\}}$ is a $(\mathcal{P}(X) - \{\varnothing\})$-indexed set, which we can write in sequence notation as $id_{\mathcal{P}(X)-\{\varnothing\}} = \{A\}_{A\in\mathcal{P}(X)-\{\varnothing\}}$. Of course, since for each $A \in \mathcal{P}(X) - \{\varnothing\}$ the set A is nonempty, by AC the set $\bigtimes id_{\mathcal{P}(X)-\{\varnothing\}} = \bigtimes_{A\in\mathcal{P}(X)-\{\varnothing\}} A$ is nonempty. But since $\bigcup(\mathcal{P}(X) - \{\varnothing\}) = X$, the elements of $\bigtimes_{A\in\mathcal{P}(X)-\{\varnothing\}} A$ are precisely the functions $c : \mathcal{P}(X) - \{\varnothing\} \longrightarrow X$ such that for all $A \in \mathcal{P}(X) - \{\varnothing\}$ we have $c(A) \in A$, that is, the "choice functions" in the more restricted sense of AC', and they exist by the nonemptyness of $\bigtimes_{A\in\mathcal{P}(X)-\{\varnothing\}} A$, as desired.

For the ($\Leftarrow$) direction, by Theorem 11 it is enough to show that AC' implies that any surjective $f : A \twoheadrightarrow B$ is a right inverse. But since f is surjective, we have $\{f^{-1}[\{b\}] \mid b \in B\} \subseteq \mathcal{P}(A) - \{\varnothing\}$. Let $c : \mathcal{P}(A) - \{\varnothing\} \longrightarrow A$ be such that for all $C \in \mathcal{P}(A) - \{\varnothing\}$ we have $c(C) \in C$. Then obviously the composed function

$$B \xrightarrow{\{\_\}_B} \mathcal{P}(B) \xrightarrow{f^{-1}[\_]} \mathcal{P}(A) - \{\varnothing\} \xrightarrow{c} A$$

is a left inverse to f, and therefore f is a right inverse, as desired. □

We can use the axiom of choice to prove that for any infinite set A, $\mathbb{N} \preceq A$. This agrees with our intuition that countable infinite sets are the smallest infinite sets possible, but this intuition requires a proof.

Theorem 13 *For any infinite set A, $\mathbb{N} \preceq A$.*

Proof. All we need to do is to define an injective function $f : \mathbb{N} \rightarrowtail A$. The intuitive idea of how to define such a function is a follows. We pick an element $a_0 \in A$ and define $f(0) = a_0$, then we pick $a_1 \in A - \{a_0\}$ and define $f(1) = a_1$. Likewise, we pick $a_{s(n)} \in A - \{a_0, \ldots, a_n\}$ and define $f(s(n)) = a_{s(n)}$, and so on. Since all the a_n are different, f is injective and we are done. However, this intuitive argument, although essentially sound, makes implicit use of both simple recursion and AC, and therefore needs to be developed into a *complete* proof. To give such a full proof we reason as follows. We let $\mathcal{P}_\infty(A) \subseteq \mathcal{P}(A) - \{\varnothing\}$ denote the subset of all the infinite subsets of A, which is trivially nonempty since $A \in \mathcal{P}_\infty(A)$. We then use a choice function $c : \mathcal{P}(A) - \{\varnothing\} \longrightarrow A$ to define a function

$$smaller : \mathcal{P}_\infty(A) \longrightarrow \mathcal{P}_\infty(A) : X \mapsto X - \{c(X)\}$$

which is well-defined, since if X is an infinite set, $X - \{c(X)\}$ is also infinite (otherwise X would be finite!). By simple recursion this then defines a function $rec(smaller, A) : \mathbb{N} \longrightarrow \mathcal{P}_\infty(A)$. We then define our desired $f : \mathbb{N} \rightarrowtail A : n \mapsto c(rec(smaller, A)(n))$. To see that f is injective, first observe that: (i) if $n \geq m$, then $rec(smaller, A)(n) \subseteq rec(smaller, A)(m)$, and (ii) for each $n \in \mathbb{N}$, $f(n) \in rec(smaller, A)(n)$, and $f(n) \notin rec(smaller, A)(s(n))$. Let now $n, m \in \mathbb{N}$, with $n \neq m$. Without loss of generality we may assume $n > m$. Therefore, $f(m) \notin rec(smaller, A)(s(m))$. Since $n \geq s(m)$, $rec(smaller, A)(n) \subseteq rec(smaller, A)(s(m))$; thus , $f(m) \notin rec(smaller, A)(n)$. But $f(n) \in rec(smaller, A)(n)$. Hence, $f(n) \neq f(m)$, as desired. □

Corollary 2 *(Infinite iff Dedekind-Infinite) A is infinite iff there is a proper subset $B \subset A$ such that $A \cong B$.*

Proof. For the ($\Leftarrow$) part, just note that if there is a proper subset $B \subset A$ such that $B \cong A$, A cannot be finite; therefore A must be infinite. For the ($\Rightarrow$) part, by the above theorem there is an injective function $f : \mathbb{N} \rightarrowtail A$. Let $B = A - \{f(0)\}$. The desired bijection $A \cong B$ is then given by the function $\lambda a \in A.$ **if** $a \in A - f[\mathbb{N}]$ **then** a **else** $f(s(f^{-1}(a)))$ **fi**. □

The axiom of choice has many other consequences, some rather technical and many far-reaching. Let me mention one that is very intuitive. In fact, so intuitive that one might naively think it obvious. It is called *cardinal comparability* (CC) and has an extremely simple formulation.

(CC) *Given any two sets A and B, either $A \preceq B$ or $B \preceq A$.*

Yet another, extremely useful property, equivalent to AC, is the property WO stating that every set can be *well-ordered*. We will discuss well orderings and both CC and WO in §18.

Besides its non-constructive nature, another reason why AC has a special status within ZFC is that it is *independent* of the other axioms of ZF in the following precise sense: (i) no contradiction is created by adding AC to ZF; and (ii) no contradiction is created by adding $\neg AC$ to ZF. That is, either AC or $\neg AC$ (but obviously not both!) can be added to ZF without contradiction. The proof of (i) goes back to to Göedel [20]; and that of (ii) to Fraenkel (see [52], 284–289). In fact, Fraenkel's proof of (ii) can be viewed

as a formalization of Russel's thought experiment (Rimmelda's story), since it is achieved by adding to a standard model of set theory a countable family of cardinality-2 sets, just like the family $Stockings = \{Stockings_n\}_{n\in\mathbb{N}}$.

AC was first proposed by Zermelo in 1904, together with a proof that $AC \Rightarrow WO$ (see [52], 139–141). This caused quite a stir due to its consequences (for example, the well-ordering of the reals) and its non-constructive nature. A subsequent more detailed proof of the same result by Zermelo (see [52], 183–198), is very much worth reading even today, because it addresses very cogently the various criticisms that had been leveled against his ideas and explains why and how *AC* is often used in mathematical practice. For additional discussion on the axiom of choice, its wide use in mathematics, Zorn's Lemma, and the so-called Banach-Tarski Paradox see [28].

Exercise 95 *Prove that AC is equivalent to the property that for every total relation $R : A \Longrightarrow B$ there exists a function $f : A \longrightarrow B$ such that $f \subseteq R$.*

Exercise 96 *Prove (without using AC) that if I is a finite set and $A = \{A_i\}_{i\in I}$ is an I-indexed set such that for each $i \in I$, $A_i \neq \emptyset$, then $\bigtimes A \neq \emptyset$. Therefore, AC is only needed for I-indexed sets such that I is* infinite.

Exercise 97 *Prove (using AC) that for any I-indexed set $A = \{A_i\}_{i\in I}$ such that for each $i \in I$, $A_i \neq \emptyset$, the projection functions $p_i : \bigtimes A \longrightarrow A_i$ are surjective for all $i \in I$.*

Exercise 98 *In the proof of Theorem 13, the set $\mathcal{P}_\infty(A)$ of infinite subsets of A was described informally. Give a set theory formula φ such that $\mathcal{P}_\infty(A)$ can be formally defined as the set $\mathcal{P}_\infty(A) = \{X \in \mathcal{P}(A) \mid \varphi\}$ using the (*Sep*) axiom.*

Exercise 99 *Prove that the model $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\})$ defined in §5.4 satisfies the AC axiom. (Hint: you may find it helpful to use Theorem 11).*

Exercise 100 *(The Infinite Hats Puzzle, adapted from [53]). There is a prison with a countably infinite number of prisoners, each with a prisoner number, $0, 1, 2, \ldots, n, \ldots$ on his uniform. They have unlimited mathematical powers: they can remember mathematical objects of arbitrary cardinality and can compute (in the broad sense of the word) with such objects. Also, they can survey their infinite number of fellow prisoners when they see them, and they can talk simultaneously with all of them when in the prison yard. One day the prison guards tell them:*

> *"Tomorrow we will place a hat on the head of each of you. The hat will be either black or white; but you will not be able to see it; however, you will be able to see the hats that all other prisoners wear, but you will not be able to talk to them. We will give each of you a piece of paper and a pencil where you should write your guess for the color of your own hat. If all of you, except for a finite number, guess correctly the color of your hat, all of you will be freed; otherwise, all of you will be executed. Today, we will let you talk to each other for an hour in the prison yard to try to find a strategy to win in this game."*

Can the prisoners agree ahead of time on a strategy that will make them free when the following day the guards place the hats on their heads?

Solve the same puzzle when the hats the guards place on the prisoners' heads can have any shade of grey between totally black and totally white (a shade can be represented as a real number s between 0 (black) and 1 (white)). In this case the prisoners also have infinite precision vision: *they can estimate the exact shade s of grayness of any other prisoner's hat just by looking.*

(Hint: you may find Exercise 70 useful).

Chapter 14

Well-Founded Relations, and Well-Founded Induction and Recursion

A well-founded relation on a set is, by definition, a terminating relation. Such relations have an enormous importance in both computer science and in mathematics for at least three reasons: (i) they are of course crucial for termination arguments; (ii) they vastly generalize induction on the natural numbers to induction on any well-founded relation; and (iii) they vastly generalize simple and primitive recursion to *well-founded recursion*, a very powerful and general form of recursion that is always guaranteed to terminate.

14.1 Well-Founded Relations

Definition 23 *Given a binary relation $R \subseteq A \times A$, the pair (A, R) is called a* well-founded relation *on A iff R is terminating, that is, there is no sequence $a : \mathbb{N} \longrightarrow A$ such that for each $n \in \mathbb{N}$ we have $a_n \, R \, a_{s(n)}$, or, pictorially, no sequence such that*

$$a_0 \, R \, a_1 \, R \, a_2 \; \ldots \; a_n \, R \, a_{s(n)} \; \ldots$$

Let us consider some examples. The set $\mathbb{N}$ can be endowed with several well founded relations, including the following:

- The predecessor relation (in fact, partial function) p, with $n \, p \, m \;\Leftrightarrow\; n = s(m)$.
- The strict order relation $n > m$.
- The reverse divisibility relation $n \, vid \, m$, where, by definition, $n \, vid \, m \;\Leftrightarrow\; n \neq 0 \wedge (\exists k \in \mathbb{N}) \; k \neq 1 \wedge n = k * m$.

Likewise, for any set A the set $List(A) = \bigcup_{n \in \mathbb{N}} [n \to A]$ can be endowed with a well-founded *head-rest superlist* order $(List(A), \sqsupset)$, where, if we denote a list $a : k \longrightarrow A$ as a finite sequence $a_0 \ldots a_{k-1}$, then $a \sqsupset b$ iff $k > 1$, and either $b = a_0$, or $b = a_1 \ldots a_{k-1}$.

Given any set A, the set $\mathcal{P}_{fin}(A)$ of finite subsets of A can be endowed, among others, with two different well-founded relations. The most obvious one is $(\mathcal{P}_{fin}(A), \supset)$. Another interesting well-founded relation (indeed, a subrelation of $\supset$) is the *singleton-rest* relation $(\mathcal{P}_{fin}(A), \sqsupset)$, which is analogous to the head-rest superlist relation for lists, where, by definition, given $X, Y \in \mathcal{P}_{fin}(A)$ we have $X \sqsupset Y$ iff X has at least two elements and there exists $x \in X$ such that either $Y = \{x\}$, or $Y = X - \{x\}$. Note that the inverse relation $(\mathcal{P}_{fin}(A), \subset)$ is well-founded iff A is finite.

Yet another well-founded relation is provided by the *immediate subterm* relation between *arithmetic expressions*, which is the set[1] *Exp* of expressions that can be formed out of a countable set of variables

[1] Properly speaking, since our set theory is built *ex nihilo* out of the empty set, we would need to *encode* these expressions, for example with a Gödel encoding of expressions as natural numbers. I will not bother with this. I will instead assume that we are working in a variant of set theory in which atoms such as the symbols $+$, $-$ and $*$, and the variables, are all allowed as elements of other sets. Then we can, for example, represent the expression $(1 + (x * 0))$ as the tuple $(+, 1, (*, x, 0))$.

$x, y, x', y, \ldots$ and of smaller expressions t, t', etc., by means of the following BNF-like grammar:

$$0 \mid 1 \mid x \mid (t + t') \mid (t - t') \mid (t * t')$$

The *immediate subterm* relation $t \rhd t'$ is then defined on *Exp* by the defining equivalence $t \rhd t' \;\Leftrightarrow\; (\exists t'' \in Exp)\; t = t' + t'' \;\vee\; t = t'' + t' \;\vee\; t = t' - t'' \;\vee\; t = t'' - t' \;\vee\; t = t' * t'' \;\vee\; t = t'' * t'$.

For non-examples, note that neither $(\mathbb{N}, <)$, nor $(\mathbb{Z}, >)$, nor $(\mathbb{Z}, <)$, nor $(\mathbb{R}, >)$, nor $(\mathbb{R}, <)$ are well-founded.

14.1.1 Constructing Well-Founded Relations

It is very useful to have constructions such that, if some relations on some sets are well-founded, then other relations on other sets are also well-founded.

For any set A, the relation $(A, \emptyset)$ is well-founded. In particular, $(\emptyset, \emptyset)$ is well-founded, and for any well-founded (A, R) there is a unique relation homomorphism $\emptyset : (\emptyset, \emptyset) \longrightarrow (A, R)$.

One very easy way to determine that a relation (A, R) is well-founded is to find a relation homomorphism $f : (A, R) \longrightarrow (B, G)$ to a binary relation (B, G) which is itself well-founded. If we view (A, R) as a transition system, we can equivalently express this idea by saying that to prove that (A, R) is teminating it is enough to find a possibly simpler, terminating transition system (B, G) and a simulation map $f : (A, R) \longrightarrow (B, G)$.

Lemma 13 *Let $f : (A, R) \longrightarrow (B, G)$ be a relation homorphism with (B, G) well-founded. Then (A, R) is well-founded.*

Proof. Suppose not. Then we have a sequence $a : \mathbb{N} \longrightarrow A$ such that for each $n \in \mathbb{N}$ we have $a_n \, R \, a_{s(n)}$. But this then yields a sequence $a; f : \mathbb{N} \longrightarrow B$ such that for each $n \in \mathbb{N}$ we have $f(a_n) \, G \, f(a_{s(n)})$, contradicting the well-foundedness of (B, G). □

The following are then two immediate consequencess of the above lemma:

Corollary 3 *Let (B, G) be a well-founded relation. Then:*

1. *Any (B', G') with $B' \subseteq B$, $G' \subseteq G$, and $G' \subseteq B' \times B'$ is well-founded. In particular, for any $B' \subseteq B$, the binary relation $(B', G \cap B'^2)$, called the* restriction *of G to B', and denoted $(B', G|_{B'})$ is well-founded.*

2. *For any function $f : A \longrightarrow B$, the binary relation $(A, f^{-1}(G))$ is well-founded, where, by definition, $a \, f^{-1}(G) \, a' \;\Leftrightarrow\; f(a) \, G \, f(a')$.*

Another useful construction is the *cartesian product*. Given well-founded relations (A, R) and (B, G), it is trivial to show that the relation $(A \times B, R \times G)$ is well-founded, where, by definition, $(a, b) \, R \times G \, (a', b') \;\Leftrightarrow\; a \, R a' \;\wedge\; b \, G \, b'$.

A second, also well-founded, relation $(A \times B, R \bowtie G)$ can be defined on $A \times B$, where, by definition, $(a, b) \, R \bowtie G \, (a', b') \;\Leftrightarrow\; (a \, R \, a' \;\wedge (b = b' \;\vee\; b \, G \, b')) \vee ((a = a' \;\vee\; a \, R \, a') \;\wedge\; b \, G \, b')$.

Yet a third, well-founded, *lexicographic relation* $(A \times B, Lex(R, G))$ can be defined on $A \times B$, where, by definition, $(a, b) \, Lex(R, G) \, (a', b') \;\Leftrightarrow\; a \, R \, a' \;\vee\; (a = a' \;\wedge\; b \, G \, b')$.

Exercise 101 *Generalize the above cartesian product construction to the product of an I-indexed family of well-founded relations $\{(A_i, R_i)\}_{i \in I}$, by showing that the binary relation $(\bigtimes_{i \in I} A_i, \bigtimes_{i \in I} R_i)$ is well-founded, where, by definition, for $a, a' \in \bigtimes_{i \in I} A_i$ we have $a \bigtimes_{i \in I} R_i \, a' \;\Leftrightarrow\; (\forall i \in I) \; a(i) \, R_i \, a'(i)$. Specialize this construction to prove that, given a well-founded relation (A, R) and a set B, the function set $[B \to A]$ is well-founded by means of the relation $f \, [B \to R] \, g \;\Leftrightarrow\; (\forall b \in B) \; f(b) \, R \, g(b)$, where $[B \to R]$ is a* purely formal notation*, which does not denote a function set but a relation between functions.*

Dually, it is trivial to show that the *disjoint union* of two well-founded relations (A, R) and (B, G) yields a well-founded relation $(A \oplus B, R \oplus G)$, where, by definition, $(x, i) \, R \oplus G \, (y, j) \;\Leftrightarrow\; (i = j = 0 \;\wedge\; x R y) \;\vee\; (i = j = 1 \;\wedge\; x G y)$.

Exercise 102 *Generalize the above disjoint union construction to the disjoint union of an I-indexed family of well-founded sets $\{(A_i, R_i)\}_{i \in I}$, by showing that the binary relation $(\bigoplus_{i \in I} A_i, \bigoplus_{i \in I} R_i)$ is well-founded, where, by definition, for $(a, i), (a', i') \in \bigoplus_{i \in I} A_i$ we have $(a, i) \bigoplus_{i \in I} R_i \, (a', i') \;\Leftrightarrow\; i = i' \;\wedge\; a \, R_i \, a'$. Specialize this construction to prove that, given a well-founded set (A, R) and another set B, the cartesian product $A \times B$ is well-founded with the relation $(a, b) \, R \times B \, (a, b') \;\Leftrightarrow\; a \, R \, a' \;\wedge\; b = b'$.*

Yet another very useful construction is to associate to a well-founded relation (A, R) the well founded relation (A, R^+). Indeed, the following lemma has a trivial proof that is left as an exercise.

Lemma 14 *If (A, R) is well founded and transitive, then it is a (strict) poset. For any well-founded (A, R) the relation (A, R^+) is also well-founded and of course a strict poset. Furthermore, any relation (A, R) is well-founded iff (A, R^+) is well-founded.*

As an example, note that $(\mathbb{N}, >) = (\mathbb{N}, p^+)$.

14.2 Well-Founded Induction

Given a binary relation (A, R), we call an element $a \in A$ *R-minimal* iff $R[\{a\}] = \emptyset$. The notion of R-minimality provides a very useful, alternative characterization of well-founded relations that makes a crucial use of the axiom of choice.

Theorem 14 *Given a binary relation (A, R) the following are equivalent:*

1. *(A, R) is well-founded.*

2. *Any nonempty subset $B \subseteq A$ has an $R|_B$-minimal element.*

Proof. To see (2) $\Rightarrow$ (1), assume that (A, R) satisfies (2) but is not well-founded. Then consider a sequence $a : \mathbb{N} \longrightarrow A$ such that for each $n \in \mathbb{N}$ we have $a_n \, R \, a_{s(n)}$. The set $B = \{a_n \mid n \in \mathbb{N}\}$ is obviously nonempty, but it has no $R|_B$-minimal element, contradicting assumption (2).

To see (1) $\Rightarrow$ (2), assume that (A, R) is well-founded and has a nonempty subset $B \subseteq A$ with no $R|_B$-minimal element. This exactly means that for each $x \in B$ the set $R[\{x\}] \cap B \in \mathcal{P}(B)$ is nonempty. But by (AC') we know that there is a function $c : \mathcal{P}(B) - \{\emptyset\} \longrightarrow B$ such that for each $X \in \mathcal{P}(B) - \{\emptyset\}$ we have $c(X) \in X$. This means that we can define a function $next = \lambda x \in B.\ c(R[\{x\}] \cap B) \in B$, where, by construction, $x \, R \, c(R[\{x\}] \cap B)$. Since B is nonempty, let $b \in B$, and consider the simply recursive sequence $rec(next, b) : \mathbb{N} \longrightarrow B$. By construction this sequence is such that for each $n \in \mathbb{N}$ we have $rec(next, b)(n) \; R \; rec(next, b)(s(n))$, contradicting (1). $\square$

As an immediate collorary of the above theorem we now obtain an enormously general induction principle, namely, the principle of well-founded induction.

Theorem 15 *(Well-Founded Induction). For any well founded (A, R), if a subset $B \subseteq A$ is such that for each $a \in A$ if $R[\{a\}] \subseteq B$ then $a \in B$, then we must have $B = A$.*

Proof. Let $B \subseteq A$ be such that for each $a \in A$ if $R[\{a\}] \subseteq B$ then $a \in B$, and assume $B \neq A$. Then $A - B$ is nonempty and has an $R|_{(A-B)}$-minimal element, say a. Therefore, $R[\{a\}] \subseteq B$. Therefore, $a \in B$, contradicting $a \in A - B$. $\square$

Given a well-founded set (A, R), we often use the above well-founded induction principle to prove formulas of the form $(\forall x \in A)\ P(x)$. Any such formula determines a subset $B = \{x \in A \mid P(x)\}$. Obviously, $(\forall x \in A)\ P(x)$ holds iff $B = A$. Using well-founded induction we can prove $B = A$, and therefore our formula, if we can prove

$$(\forall a \in A)\ [(\forall x \in R[\{a\}])\ P(x)] \Rightarrow P(a).$$

The above induction principle is very general, and contains many other induction principles as special cases, including the following (check in detail that the mentioned induction principles are indeed special cases as claimed):

- **Peano Induction**. This is just the special case of well-founded induction where $(A, R) = (\mathbb{N}, p)$.

- **Strong Induction**. This is is a variant of induction on the natural numbers where for a set $B \subseteq \mathbb{N}$, if we can show $0 \in B$, and that for each $n \in \mathbb{N}$ whenever $n \subseteq B$ then $n \in B$, then we can conclude that $B = \mathbb{N}$. This is just the special case of well-founded induction where $(A, R) = (\mathbb{N}, >)$.

- **List Induction**. To prove a property P for all lists in $List(A)$ it is enough to show that: (i) $P(\emptyset)$, (ii) $(\forall a \in A)\, P(a)$ (where we view each $a \in A$ as a lenght-one list), and (iii) for any list $a_0 \dots a_{k-1}$ with $k > 1$ we can prove that $P(a_0)$ and $P(a_1 \dots a_{k-1})$ imply $P(a_0 \dots a_{k-1})$. This is just the special case of well-founded induction where $(A, R) = (List(A), \sqsupset)$.

- **Structural Induction**. To prove a property P for all expressions in an algebraic language (arithmetic expressions Exp are a paradigmatic example, but we could consider expressions built with any other function symbols) we: (i) prove $P(a)$ for each constant symbol a; (ii) prove $P(x)$ for each variable x if the expressions have variables; and (iii) for each function symbol f of n arguments in our expression language and for each expression $f(t_1, \dots, t_n)$ we prove that if $P(t_1), \wedge \dots \wedge P(t_n)$, then $P(f(t_1, \dots, t_n))$. For our arithmetic expression example this means that to prove that P holds for all expressions we have to prove: $P(0)$, $P(1)$, $P(x)$ for all variables x, and for $f \in \{+, -, *\}$ that if $P(t)$ and $P(t')$, then $P(f(t, t'))$. This is just well-founded induction where $(A, R) = (Exp, \rhd)$ (the generalization to expressions with other function symbols is straightforward).

Note that the "base case" is *implicit* in the general formulation of well-founded induction. For the natural numbers the base case is the case $n = 0$. For a well-founded (A, R), any R-minimal $a \in A$ provides a base case, since then $R[\{a\}] = \emptyset$, and for $B = \{x \in A \mid P(x)\}$ we then obviously have $\emptyset = R[\{a\}] \subseteq B$; therefore we must prove $P(a)$ for all R-minimal $a \in A$. For example, for $(List(A), \sqsupset)$, the $\sqsupset$-minimal lists are the empty list $\emptyset$ and the length-one lists $a \in A$, which provide the base cases for list induction. Therefore, to inductively prove a property P for all lists, in particular we must prove $P(\emptyset)$ and $P(a)$ for each $a \in A$.

14.3 Well-Founded Recursion

The idea of well founded recursion is as follows. We have a well-founded set (A, R) and another set B. We then define a recursive function $f : A \longrightarrow B$ by defining the value of f for any argument $a \in A$ in terms of the value of f for "R-smaller" arguments, that is, for arguments in $R[\{a\}]$. Let us see some examples of this very general method to define functions before we formalize the general notion.

14.3.1 Examples of Well-Founded Recursion

Simple recursion is a special case of this idea. We take $(A, R) = (\mathbb{N}, p)$. Then, given $b \in B$ and $f : B \longrightarrow B$, we define $rec(f, b) : \mathbb{N} \longrightarrow B$ in terms of the values of $rec(f, a)$ for "p-smaller" numbers. The base case is $n = 0$, which has nothing p-smaller than it. So we set $rec(f, b)(0) = b$. In all other cases, $n = s(m)$, so that m is the element p-smaller than $s(m)$, and we set $rec(f, b)(s(m)) = f(rec(f, b)(m))$.
Primitive recursion is another special case of this idea. We take $(A, R) = (\mathbb{N} \times B, p \times B)$ (see Exercise 102). Given $g : B \longrightarrow A$ and $f : A \longrightarrow A$, we define $prec(f, g) : \mathbb{N} \times B \longrightarrow A$ in terms of its values for $p \times B$-smaller arguments. The $p \times B$-minimal elements are those of the form $(0, b)$, for which we define $prec(f, g)(0, b) = g(b)$. All other elements are of the form $(s(n), b)$, whose only $p \times B$-smaller element is (n, b), so we define $prec(f, g)(s(n), b) = f(prec(f, g)(n, b))$.
Ackermann's Function can also be defined by well-founded recursion. We take $(A, R) = (\mathbb{N}\times\mathbb{N}, Lex(p, p))$. $A : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ is then defined in terms of its values on $Lex(p, p)$-smaller arguments. For $(0, n) \in \mathbb{N} \times \mathbb{N}$ we define $A(0, n) = s(n)$. For $(s(m), 0) \in \mathbb{N} \times \mathbb{N}$, the $Lex(p, p)$-smaller arguments are all the (m, n), and we define $A(s(m), 0) = A(m, 1)$. Finally, for $(s(m), s(n)) \in \mathbb{N} \times \mathbb{N}$, the $Lex(p, p)$-smaller arguments are all the (m, k), plus the element $(s(m), n)$, so we can define $A(s(m), s(n)) = A(m, A(s(m), n))$.
List Recursion works similarly to primitive recursion. Here, the well-founded relation is $(List(A), \sqsupset)$, and the $\sqsupset$-minimal lists are $\emptyset$ and the $a \in A$. So, to define a recursive function, let us call it $rec(f, g, b) : List(A) \longrightarrow B$, we need to specify an element $b \in B$ and functions $g : A \longrightarrow B$ and $f : B\times B \longrightarrow B$. Then we define $rec(f, g, b)$ as follows: (i) $rec(f, g, b)(\emptyset) = b$, (ii) for $a \in A$, $rec(f, g, b)(a) = g(a)$, and (iii) for a list $a_0 \dots a_{k-1}$ with $k > 1$, $rec(f, g, b)(a_0 \dots a_{k-1}) = f(g(a), rec(f, g, b)(a_1 \dots a_{k-1}))$. For example, suppose that $B = List(C)$, and that we want to extend a function $g : A \longrightarrow C$ to a function $map(g) : List(A) \longrightarrow List(C)$ in the obvious way, that is: (i) $map(g)(\emptyset) = \emptyset$, (ii) for $a \in A$, $map(g)(a) = g(a)$, and (iii) for a list $a_0 \dots a_{k-1}$ with $k > 1$, $map(g)(a_0 \dots a_{k-1}) = g(a)\, map(g)(a_1 \dots a_{k-1})$. That is, $map(g) = rec(\_ \cdot \_, g, \emptyset)$, where $\_ \cdot \_ : List(C) \times List(C) \longrightarrow List(C) : (a_0 \dots a_{n-1}, b_0 \dots b_{m-1}) \mapsto a_0 \dots a_{n-1}\, b_0 \dots b_{m-1}$.

Structural Recursion can be used to define recursive functions on algebraic expressions. We can illustrate the idea for the case *Exp* of arithmetic expressions, but all generalizes to expressions with other function symbols in a straightforward way. The well-founded relation is of course $(Exp, \rhd)$. To define a function from *Exp* to B by structural recursion we need to specify: (i) elements $b_0, b_1 \in B$, (ii) a function $g : \mathscr{V} \longrightarrow B$, where $\mathscr{V}$ is the set of variables in *Exp*, and (iii) binary functions $f_+, f_-, f_* \in [B{\times}B{\rightarrow}B]$. Then we can define a function, let us call it $rec(f, g, b) : Exp \longrightarrow B$, as follows: (i) $rec(f, g, b)(0) = b_0$, and $rec(f, g, b)(1) = b_1$, (ii) $rec(f, g, b)(x) = g(x)$, and (iii) for $op \in \{+, -, *\}$, $rec(f, g, b)(op(t, t')) = f_{op}(rec(f, g, b)(t), rec(f, g, b)(t'))$. For example, suppose that *Exp* are arithmetic expressions in a programming language, and that the variables appearing in the expressions have some assigned integer values in memory (if some variables are unasigned, we may by default give them the value 0). This means that the state of the memory can be characterized by a function $g : \mathscr{V} \longrightarrow \mathbb{Z}$. Then the evaluation function for arithmetic expressions associated to the memory g is the function $eval(g) : Exp \longrightarrow \mathbb{Z}$ that in the above notation can be defined as $eval(g) = rec(f, g, b)$ with $b_0 = 0$, $b_1 = 1$, and f_+, f_-, f_* the functions $+, -, * : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$. For example, if $g(x) = 3$ and $g(y) = 2$, then $eval(g)(x * (x + y)) = 15$.

14.3.2 Well-Founded Recursive Definitions: Step Functions

The above examples give us a good, informal *intuition* for what a well-founded recursive function looks like. But they do *not* give us a *mathematical model* for well-founded recursive functions. Without such a mathematical model we cannot answer crucial questions such as the following: (i) what is a well-founded recursive function definition? (ii) how do we *know* that each such definition indeed defines a unique function and therefore makes sense? (iii) how do we *know* that all functions defined by well-founded recursion terminate? This section answers question (i) by developing a very general mathematical model of well-founded recursive definitions. Questions (ii) and (iii) are then answered in §14.3.3.

In several of the examples of well-founded recursive function presented in §14.3.1 we used some *linguistic description* of the process of defining the value of a well-founded recursive function on an argument in terms of its value on R-smaller arguments. But since there isn't a single well-founded relation, but an *infinite* number of such relations, what we need, and we do not yet have, is a precise formulation of a *general method* of *defining* well-founded recursive functions that will apply to *any* well founded relation. Furthermore, this general method should be *semantic*, that is, involving sets and functions, as opposed to linguisitc. Indeed, it should provide the semantic basis to properly interpret any linguistic description, such as, for example, a well-founded recursive program.

Such a semantic method of defining well-founded recursive functions is provided by the notion of a *step function*, which is a function that contains all the information necessary to allow us to take the next step in computing our desired function. Let me first illustrate this idea with an example before giving the general definition. Consider the following primitive recursive definition of the addition function by means of the equalities:

- $0 + m = m$
- $s(n) + m = s(n + m)$.

This is indeed a linguistic description. But what does it really say? It is obviously *circular*, since + is defined in terms of itself. But this circular syntactic description implicitly contains a *non-circular* semantic method to *extend* a function, so that it becomes defined on a bigger argument. It says, "if you have any two-argument function f on natural numbers which is defined on smaller arguments, I can tell you how to extend it to a bigger argument." For example, if f is defined on (n, m), then we can extend f to be defined on the bigger argument $(s(n), m)$ with the value $f(s(n), m) = s(f(n, m))$.

How can we precisely capture this non-circular semantic method? With a function. Which function? In this example, the function that maps each pair $(f, (n, m))$ with f any function defined on the set $R[\{(n, m)\}]$ into the intended value of its extension for the argument (n, m). Recall that in this case R is the well-founded relation $p{\times}\mathbb{N}$. Therefore, $R[\{(0, m)\}] = \emptyset$, and $R[\{(s(n), m)\}] = \{(n, m)\}$. The function in question can then be described by the lambda expression $\lambda(f, (n.m))$. **if** $n = 0$ **then** m **else** $s(f(p(n), m)$ **fi**. What are its domain and codomain? Its codomain is obviously $\mathbb{N}$. What about its domain? Here our knowldge of I-indexed sets becomes extremely useful, since we can say it in one blow: its domain is the set $\bigoplus_{(n,m)\in\mathbb{N}\times\mathbb{N}}[R[\{(n, m)\}]{\rightarrow}\mathbb{N}]$, where, as already mentioned, $R = p \times \mathbb{N}$. Indeed, the elements of such a disjoint union are *exactly* the pairs

of the form $(f, (n.m))$ with f a function with domain $R[\{(n, m)\}]$ and codomain $\mathbb{N}$. This is just a special instance of the following general notion.

Definition 24 *Let (A, R) be a well-founded relation and B any set. A* definition *of a well-founded recursive function from A to B is a function of the form $h : \bigoplus_{a \in A}[R[\{a\}] \rightarrow B] \longrightarrow B$, called the* step function *of the recursive function thus defined.*

In what sense does a step function *define* a well-founded recursive function? In the following: a step function $h : \bigoplus_{a \in A}[R[\{a\}] \rightarrow B] \longrightarrow B$ defines the recursive function, let us call it $rec(h)$, characterized by means of the defining equality

$$rec(h)(x) = h(rec(h)\restriction_{R[\{x\}]}, x).$$

Of course this is still a *circular* definition, but now a semantic one in terms of h, and not by some linguistic device. However, it is not at all obvious that the above equality actually defines a *unique* function, that such a function exists at all, or that it is terminating, although intuitively all these things seem *plausible*. Settling these burning issues once and forall is the goal of §14.3.3.

For the moment we can gain further empirical evidence that defining $rec(h)$ in this way makes sense by observing that the step function h provides a *computational method* to compute $rec(h)(x)$. Let us illustrate this idea with our primitive recursive definition of addition by seeing how we can compute that $2 + 2 = 4$. Our function h in this case is $h = \lambda(f, (n.m))$. **if** $n = 0$ **then** m **else** $s(f(p(n), m)$ **fi**. Therefore, $2 + 2 = rec(h)(2, 2) = h(rec(h) \restriction_{\{(1,2)\}}, (2, 2)) = s(rec(h)(1, 2)) = s(h(rec(h) \restriction_{\{(0,2)\}}, (1, 2))) = s(s(rec(h)(0, 2))) = s(s(2)) = 4$.

To gain a little more familiarity with step functions, we can see what a step function looks like for a type of list-recursive function considered in §14.3.1, namely, a function of the form $map(f)$; specifically one such function that allows us to compute the *lenght* of a list. The idea is to define a function *lenght* on all elements of $List(A)$ as the function $length = map(!_A) : List(A) \longrightarrow List(\{1\})$, where $!_A$ is the constant function mapping each $a \in A$ to 1. Note that the set of lists $List(\{1\})$, with the empty list $\emptyset$ as *zero* element, and the function $\lambda l \in List(\{1\}).\ 1\, l$, that appends one more "1" to the left of a list l, as the *successor* function, satisfies the Peano-Lawvere axiom, and therefore is a model of the natural numbers, namely, the model corresponding to "counting with one's fingers," so that, for example, the number 5 is represented by the list 1 1 1 1 1. Our step function for the above length function is then of the form $h : \bigoplus_{l \in List(A)}[\sqsupset [\{l\}] \rightarrow List(\{1\})] \longrightarrow List(\{1\})$, and has the following definition by cases (which could easily be expressed by a nested if-then-else):

- $h(\emptyset, \emptyset) = \emptyset$
- $(\forall a \in A)\ h(\emptyset, a) = 1$
- for $k > 1$, $h(f, a_0 \ldots a_{k-1}) = 1\, f(a_1 \ldots a_{k-1})$.

Suppose now that $a, b \in A$ and let us compute $length(a\, b\, a\, b)$. We have, $length(a\, b\, a\, b) = rec(h)(a\, b\, a\, b) = h(rec(h) \restriction_{\{a,b\, a\, b\}}, (a\, b\, a\, b)) = 1\, rec(h)(b\, a\, b) = 1\, h(rec(h) \restriction_{\{b,a\, b\}}, (b\, a\, b)) = 1\, 1\, rec(h)(a\, b) = 1\, 1\, h(rec(h) \restriction_{\{a,b\}}, (a\, b)) = 1\, 1\, 1\, rec(h)(b) = 1\, 1\, 1\, h(\emptyset, b) = 1\, 1\, 1\, 1$.

Exercise 103 *Define the appropriate step function h for Ackermann's funtion and for the function $eval(g) : Exp \longrightarrow \mathbb{Z}$ evaluating arithmetic expressions with a memory g. Use the corresponding step functions to compute the value of each of these functions on some small arguments.*

Exercise 104 *(Alternative definition of step functions). Given a well-founded relation (A, R) prove that there is a containment $\bigoplus_{a \in A}[R[\{a\}] \rightarrow B] \subseteq [A \rightarrow B] \times A$. Show that, instead of using a step function $h : \bigoplus_{a \in A}[R[\{a\}] \rightarrow B] \longrightarrow B$ to define a well-founded recursive function $rec(h)$, we can always extend such a function h to a function $h' : [A \rightarrow B] \times A \longrightarrow B$, so that $h' \restriction_{\bigoplus_{a \in A}[R[\{a\}] \rightarrow B]} = h$, and can then equivalently define $rec(h)$ as $rec(h')$ by the equality $rec(h')(x) = h'(rec(h') \restriction_{R[\{x\}]}, x)$. Therefore, we could alternatively have defined a step function as a function $h' : [A \rightarrow B] \times A \longrightarrow B$.*

14.3.3 The Well-Founded Recursion Theorem

Let us come back to the three questions raised in §14.3.2: (i) what is a well-founded recursive function definition? (ii) how do we *know* that each such definition indeed defines a unique function and therefore makes sense? (iii) how do we *know* that all functions defined by well-founded recursion terminate? The answer to question (i) is now clear: a step function. The following theorem answers questions (ii) and (iii). Note that the proof of this theorem is a natural, yet far-reaching, generalization of the same result, in Theorem 5, for the special case of simple recursion, that is, of well-founded recursion for the well-founded relation $(\mathbb{N}, p)$.

Theorem 16 *(Well-Founded Recursion). Let (A, R) be a well-founded relation, and consider a step function $h : \bigoplus_{a \in A}[R[\{a\}] \to B] \longrightarrow B$. Then there exists a unique function $rec(h) : A \longrightarrow B$, such that for all $x \in A$, the function $rec(h)$ satisfies the equality $rec(h)(x) = h(rec(h)\restriction_{R[\{x\}]}, x)$.*

Proof. We first prove that such a function exists, and then that it is unique. Let $\mathcal{R}(h)$ be the set of all *h-closed* relations, where, by definition, a relation $Q \subseteq A \times B$ is h-closed iff for each $x \in A$, if there is a function $g : R[\{x\}] \longrightarrow B$ such that $g \subseteq Q$, then we have $(x, h(x, g)) \in Q$. Since $A \times B \in \mathcal{R}(h)$, the set $\mathcal{R}(h)$ is nonempty. Define $rec(h) = \bigcap \mathcal{R}(h)$. It is then trivial to check that $rec(h) \in \mathcal{R}(h)$. We will be done with the existence part if we prove: (i) $rec(h)$ is *total*, that is, $(\forall x \in A)(\exists y \in B)\ (x, y) \in rec(h)$; and (ii) $rec(h)$ is a function; since (i) and (ii) imply that for all $x \in A$, $rec(h)(x) = h(rec(h)\restriction_{R[\{x\}]}, x)$.

To see (i), suppose that $rec(h)$ is not total. This means that the set $X = \{x \in A \mid \neg(\exists y \in B)\ (x, y) \in rec(h)\}$ is nonempty and has a R-minimal element $a \in X$. Therefore, $rec(h)$ is total on $A - X$, and $R[\{a\}] \subseteq (A - X)$. This means that the projection function $p_1 = \lambda(x, y) \in rec(h).\ x \in (A - X)$ is surjective. Therefore, by Theorem 11, we have a function $q : (A - X) \longrightarrow rec(h)$ such that $q; p_1 = id_{A-X}$. Therefore, $g = q[A - X] \subseteq rec(h)$ is a function. Therefore, since $R[\{a\}] \subseteq (A - X)$ and $rec(h)$ is h-closed, we must have $(a, h(a, g\restriction_{R[\{a\}]}) \in rec(h)$, contradicting the fact $rec(h)$ is undefined on a. To see (ii) suppose that $rec(h)$ is not a function. This means that the set $Y = \{x \in A \mid (\exists y, z)\ (x, y), (x, z) \in rec(h) \wedge y \neq z\}$ is nonempty and has a R-minimal element $a' \in Y$. Therefore, by (i), $rec(h)$ is a function on $A - Y$, and $R[\{a'\}] \subseteq (A - Y)$. But since $rec(h)$ is h-closed, we must have $(a', h(a', rec(h)\restriction_{R[\{a'\}]})) \in rec(h)$, and since $a' \in Y$, we must have a $b \in B$ with $(a', b) \in rec(h) \wedge b \neq h(a', rec(h)\restriction_{R[\{a'\}]})$. But if we can show that $rec'(h) = rec(h) - \{(a', b)\}$ is h-closed, then we get a blatant contradiction of the fact that $rec(h)$ is the *smallest* h-closed relation. But we can see that $rec'(h)$ is indeed h-closed reasoning by cases. If $x = a'$, then since $rec'(h)\restriction_{R[\{a'\}]} = rec(h)\restriction_{R[\{a'\}]}$ is a function, there is exactly one choice for a function $g : R[\{a'\}] \longrightarrow B$ such that $g \subseteq rec'(h)$, namely, $g = rec'(h)\restriction_{R[\{a'\}]}$, and we have $(a', h(a', rec'(h)\restriction_{R[\{a'\}]})) \in rec'(h)$ by the definition of $rec'(h)$. And if $x \neq a'$, then if $g : R[\{x\}] \longrightarrow B$ is such that $g \subseteq rec'(h)$, then, a fortiori, $g \subseteq rec(h)$, and therefore $(x, h(x, g)) \in rec(h)$ by $rec(h)$ being h-closed. But since $x \neq a'$ this means that $(x, h(x, g)) \in rec'(h)$, and therefore $rec'(h)$ is h-closed, against the minimality of $rec(h)$. Therefore, by (i) and (ii), $rec(h)$ exists and is a total function $rec(h) : A \longrightarrow B$.

To prove uniqueness, suppose that there is another function, say, g, satisfying the conditions stated in the theorem. Then we must have $g \in \mathcal{R}(h)$, which forces the set-theoretic containment $rec(h) \subseteq g$. But this implies $rec(h) = g$, since, in general, given any two sets X and Y, and any two functions $f, f' \in [X \to Y]$, whenever $f \subseteq f'$ we must have $f = f'$. □

Exercise 105 *Prove that the length of the concatenation of two lists is the sum of their lengths. Since in $List(\{1\})$ the list concatenation operation $\_ \cdot \_$ becomes natural number addition, this means that, given any set A, we must show that the function $length = map(!_A) : List(A) \longrightarrow List(\{1\})$, satisfies the equality:*

$$(\forall l, l' \in List(A))\ length(l \cdot l') = length(l) \cdot length(l').$$

(Hints: (i) use well-founded induction on $(List(A), \sqsupset)$; (ii) you may be more ambitious and prove the much more general equality $(\forall l, l' \in List(A))\ map(f)(l \cdot l') = map(f)(l) \cdot map(f)(l')$ for any $f : A \longrightarrow B$, getting the result for $length = map(!_A)$ as a special case. Note that proving more general theorems is not necessarily harder: in many cases it is easier*! I call this approach "generalize and conquer").*

Chapter 15

Cardinal Numbers and Cardinal Arithmetic

Given a finite set A with n elements, we call n the *cardinality* of A, and use the notation $|A| = n$. But n is itself a set with n elements, so that we have $A \cong n$, and $||A|| = |n| = n = |A|$. Also, since there is a bijection between two finite sets A and B if and only if A and B have the same number of elements, we have:

$$A \cong B \;\Leftrightarrow\; |A| = |B|$$

This means that natural numbers give us a way of obtaining a *canonical representative set* for all finite sets of same size n, namely, the set n.

Furthermore, there is a striking similarity between arithmetic operations on numbers and corresponding operations on finite sets, since by Exercises 16 and 28, for A and B finite sets, we have the arithmetic identities:

$$\begin{aligned} |A \oplus B| &= |A| + |B| \\ |A \times B| &= |A| \cdot |B| \\ |[A \to B]| &= |B|^{|A|} \end{aligned}$$

Note that the above identities allow us to "turn arithmetic on its head," by reducing arithmetic to set theory. That is, given two natural numbers $n, m \in \mathbb{N}$ we can *define* their addition, multiplication, and exponentiation (without any previous knowledge of what those functions are) by means of the defining equations:

$$\begin{aligned} n + m &= |n \oplus m| \\ n \cdot m &= |n \times m| \\ n^m &= |[m \to n]| \end{aligned}$$

Georg Cantor generalized the idea of cardinality to *any* set. In [9], §1, he stated that for any set M its *cardinal number* $|M|$ is:

> *the general concept which, by means of our active faculty of thought, arises from the aggregate M when we make abstraction of the nature of its various elements m and of the order in which they are given.*

Cantor's meaning becomes particularly clear when we apply his words to the case when A is a *finite* set, because what our active faculty of thought arrives at when it abstracts away the particular details about the elements of a finite set A is precisely the concept of its *number* of elements. What Cantor is therefore saying is that we can perform the same kind of abstraction for any set.

In terms of *ZFC*'s axiomatic formalization of set theory, we can rephrase Cantor's claims about the existence of cardinal numbers, and the basic relationships between sets and their cardinal numbers, as the claim that there is a *definitional extension* of set theory by means of a new unary operation on sets, denoted $|\_|$, such that it satisfies the following properties (see [9], §1):

1. For A a finite set, $|A|$ is its cardinality in the usual sense of its number of elements.
2. For any set A we have $A \cong |A|$.
3. For any two sets A and B we have the equivalence $A \cong B \;\Leftrightarrow\; |A| = |B|$.

Cantor did not construct the set $|A|$, since he rather viewed $|A|$ as the abstract concept associated to what later would be called the "equivalence class" of all sets of cardinality $|A|$. Unfortunaltely, such an equivalence class is *not* a set (but see Scott's trick in §19.3). Giving an explict construction for the set $|A|$ for any A in terms of the rest of set theory —thus showing that the interpretation of the unary operation $|\_|$ is indeed obtained by a definitional extension and therefore does not change set theory at all, nor requires any extra axioms— remained an open problem for decades. It was solved in the 1920's by John von Neumann as part of his theory of ordinals. We will define von Neumann's ordinals, and will use them to give an explicit construction of cardinals as special ordinals in §18. For the moment, just knowing that the $|\_|$ operation exists and satisfies properties (1)–(3) above will allow us to learn quite a few things about cardinals and their basic properties, without any need for the details of how cardinals are constructed.

Note that it follows trivially from properties (2) and (3) that $||A|| = |A|$. Therefore, we can characterize cardinals as those sets κ such that $|\kappa| = \kappa$. In what follows, variables κ, λ, μ and their subscripted versions will range over cardinals, the same way that ordinary variables $x, y, z, X, Y, Z, A, B, C, \ldots$ range over sets in our set theory formulas. Of course, this is just for notational convenience, since, using the explicit definition of the operation $|\_|$ in §18, we can define a unary predicate *card* on sets by means of the definitional equivalence

$$card(x) \quad \Leftrightarrow \quad |x| = x$$

and can then eliminate all mention of variables κ, λ, μ in any formula. That is, $(\forall\kappa)\ \varphi(\kappa)$ can be replaced by $(\forall x)\ (card(x) \Rightarrow \varphi(x))$, and $(\exists\kappa)\ \varphi(\kappa)$ can be replaced by $(\exists x)\ (card(x) \wedge \varphi(x))$.

Note that we can define a *predicate card* in our set theory language, but not a *set*, say **Card**, of all cardinals. Cantor was aware of this paradox as early as 1899 (see [52], 113–117).

Lemma 15 *There is no set such that all cardinals κ belong to it.*

Proof. Suppose such a set, say W, were to exist. Then we could define the set of all cardinals as: $\mathbf{Card} = \{x \in W \mid x = |x|\}$. Let $Q = \bigcup \mathbf{Card}$, then for every cardinal κ we have $\kappa \subseteq Q$. In particular, $|\mathcal{P}(Q)| \subseteq Q$, which by $\mathcal{P}(Q) \cong |\mathcal{P}(Q)|$, gives us $\mathcal{P}(Q) \preceq Q$, which together with $Q \preceq \mathcal{P}(Q)$, gives us $\mathcal{P}(Q) \cong Q$ by the Schroeder-Bernstein Theorem. But this contradicts Cantor's Theorem. □

15.1 Cardinal Arithmetic

The idea of "turning arithmetic on its head" by defining all arithmetic operations in terms of their corresponding set-theoretic operations, as we have done above, can be generalized to *all* cardinals. In this way, we extend ordinary arithmetic to so-called *cardinal arithmetic* (also due to Cantor: see [9]). In fact, we can use the exact same set-theoretic definition of addition, multiplication, and exponentiation of two natural numbers n and m, to define addition, multiplication, and exponentiation of *any* two cardinals κ and λ by means of the defining equalities:

$$\begin{aligned} \kappa + \lambda &= |\kappa \oplus \lambda| \\ \kappa \cdot \lambda &= |\kappa \times \lambda| \\ \kappa^{\lambda} &= |[\lambda \to \kappa]| \end{aligned}$$

Furthermore, addition and multiplication of cardinals can be defined not just for a finite collection of cardinals, but also for any I-indexed set of cardinals. That is, if $\{\kappa_i\}_{i\in I}$ is an I-indexed set of cardinals, we can generalize binary cardinal addition and multiplication to addition and multiplication of finite or infinite families of such cardinals (depending on whether the index set I is finite or infinite) by means of the defining equalities:

$$\begin{aligned} \Sigma_{i\in I}\kappa_i &= |\bigoplus_{i\in I} \kappa_i| \\ \Pi_{i\in I}\kappa_i &= |\bigtimes_{i\in I} \kappa_i| \end{aligned}$$

How well-behaved is cardinal arithmetic? Does it satisfy any laws? It turns out that many of the laws of ordinary arithmetic extend from natural numbers to arbitrary cardinals. Furthermore, the *same* set-theoretic proofs that prove these properties for arbitrary cardinals also prove them *a fortiori* for the natural numbers. This is another consequence of "turning arithmetic on its head," namely, that many important properties of ordinary arithmetic can be given purely set-theoretic proofs. Here are some such arithmetic laws, stated for all cardinals:

Theorem 17 *For all cardinals κ, λ, μ, the following identities hold:*

$$\kappa + 0 = \kappa \tag{15.1}$$
$$\kappa + \lambda = \lambda + \kappa \tag{15.2}$$
$$(\kappa + \lambda) + \mu = \kappa + (\lambda + \mu) \tag{15.3}$$
$$\kappa \cdot 1 = \kappa \tag{15.4}$$
$$\kappa \cdot 0 = 0 \tag{15.5}$$
$$\kappa \cdot \lambda = \lambda \cdot \kappa \tag{15.6}$$
$$(\kappa \cdot \lambda) \cdot \mu = \kappa \cdot (\lambda \cdot \mu) \tag{15.7}$$
$$\kappa \cdot (\lambda + \mu) = (\kappa \cdot \lambda) + (\kappa \cdot \mu) \tag{15.8}$$
$$\kappa^0 = 1 \tag{15.9}$$
$$1^\kappa = 1 \tag{15.10}$$
$$\kappa^1 = \kappa \tag{15.11}$$
$$\kappa^{\lambda+\mu} = \kappa^\lambda \cdot \kappa^\mu \tag{15.12}$$
$$(\kappa \cdot \lambda)^\mu = \kappa^\mu \cdot \lambda^\mu \tag{15.13}$$
$$(\kappa)^{\lambda \cdot \mu} = (\kappa^\lambda)^\mu \tag{15.14}$$
$$\Sigma_{i \in I} \kappa = \kappa \cdot |I| \tag{15.15}$$
$$\Pi_{i \in I} \kappa = \kappa^{|I|} \tag{15.16}$$
$$\Sigma_{i \in I}(\kappa_i + \lambda_i) = (\Sigma_{i \in I} \kappa_i) + (\Sigma_{i \in I} \lambda_i) \tag{15.17}$$
$$\Pi_{i \in I}(\kappa_i \cdot \lambda_i) = (\Pi_{i \in I} \kappa_i) \cdot (\Pi_{i \in I} \lambda_i) \tag{15.18}$$

Proof. The proofs of these identities are all established the same way, namely, by showing that two sets have a bijection between them, which then makes their cardinals equal. For example, we can prove that $\kappa + \lambda = \lambda + \kappa$ by proving that $\kappa \oplus \lambda \cong \lambda \oplus \kappa$. We can cut our work roughly in half by reasoning by duality. That is, since $\oplus$ and $\times$, and, similarly, $\bigoplus$ and $\bigtimes$, are *dual constructions* used in the definitions of addition and multiplication, any property common to these dual operations (for example, commutativity and associativity is common to addition and multiplication) can be proved for one of the operations (say addition) and can then be dualized to prove the similar property for multiplication "for free," just by changing injections by projections and reversing the direction of all the functions used in the proof for addition.

Let me illustrate this dualization technique for the case of the commutative property. To prove it for addition, we have to show that $\kappa \oplus \lambda \cong \lambda \oplus \kappa$. We do this by showing that the function $[i'_2, i'_1] : \kappa \oplus \lambda \longrightarrow \lambda \oplus \kappa$ has the function $[i_2, i_1] : \lambda \oplus \kappa \longrightarrow \kappa \oplus \lambda$ as its inverse, where i_j, resp., i'_j, denote the corresponding injections in each case. For example, i_1 denotes the inclusion $i_1 : \kappa \rightarrowtail \kappa \oplus \lambda$, and i'_1 denotes the inclusion $i'_1 : \lambda \rightarrowtail \lambda \oplus \kappa$. Let us prove the identity $[i'_2, i'_1]; [i_2, i_1] = id_{\kappa \oplus \lambda}$ (the proof of the identity $[i_2, i_1]; [i'_2, i'_1] = id_{\lambda \oplus \kappa}$ is entirely similar). Since $id_{\kappa \oplus \lambda} = [i_1, i_2]$, by Exercise 39, to prove $[i'_2, i'_1]; [i_2, i_1] = id_{\kappa \oplus \lambda}$ is equivalent to proving the two identities $i_1; [i'_2, i'_1]; [i_2, i_1] = i_1$ and $i_2; [i'_2, i'_1]; [i_2, i_1] = i_2$. The proofs are again entirely similar. Let us prove $i_1; [i'_2, i'_1]; [i_2, i_1] = i_1$. This follows trivially, again by Exercise 39, from the sequence of identities $i_1; [i'_2, i'_1]; [i_2, i_1] = i'_2; [i_2, i_1] = i_1$.

The point now, is that, by changing the injections by projections and the direction of the functions, we also *automatically* get a dual proof (using the dual of Exercise 39, which is Exercise 38) of the fact that the function $(p_2, p_1) : \kappa \times \lambda \longrightarrow \lambda \times \kappa$ has the function $(p'_2, p'_1) : \lambda \times \kappa \longrightarrow \kappa \times \lambda$ as its inverse.

In a completely similar way, we can prove associativity of addition, getting for free a dual proof of associativity of multiplication.

The proofs of (12.12) and (12.13) are also dual. So let us prove (12.12), that is, that there is a bijection $[\lambda \oplus \mu \to \kappa] \cong ([\lambda \to \kappa] \times [\mu \to \kappa])$. This again follows easily from Exercise 39, since that exercise shows that

the mapping that sends the *pair* $(f, g) \in [\lambda{\to}\kappa] \times [\mu{\to}\kappa]$ to the function $[f, g] \in [\lambda \oplus \mu{\to}\kappa]$ is bijective, since its inverse is the mapping sending any function $h \in [\lambda \oplus \mu{\to}\kappa]$ to the pair $(i_1; h, i_2; h) \in [\lambda{\to}\kappa] \times [\mu{\to}\kappa]$.

The identities (12.9) and (12.10) are also dual of each other, since (12.9) follows trivially from the fact that $0 = \varnothing$ is the initial set, and (12.10) follows likewise trivially from the fact that 1 is the final set.

The proofs of (12.4) and (12.11) follow trivially from the respective bijections $\kappa \times 1 \cong \kappa$ and $[1{\to}\kappa] \cong \kappa$ discussed in Exercise 44. The proof of (12.5) follows trivially from the identity $\kappa \times \varnothing = \varnothing$. The proof of (12.14) follows from the bijection $curry : [\lambda \times \mu{\to}\kappa] \longrightarrow [\lambda{\to}[\mu{\to}\kappa]]$ described in Exercise 46. The proof of (12.1) follows from the identity $\kappa \oplus \varnothing = \kappa \times \{0\}$, that makes the inclusion $i_1 : \kappa \longrightarrow \kappa \oplus \varnothing$ a bijection. The proofs of (12.15) and (12.16) are both direct consequences of Exercise 90. The proofs of (12.17) and (12.18) are direct consequences of Exercise 88. □

The order on the natural numbers also extends naturally to an order on cardinals by defining for κ, λ cardinals the predicate $\kappa \leq \lambda$ by means of the equivalence

$$\kappa \leq \lambda \quad \Leftrightarrow \quad \kappa \preceq \lambda.$$

Similarly, we define the predicate $\kappa < \lambda$ by means of the equivalence

$$\kappa < \lambda \quad \Leftrightarrow \quad \kappa \prec \lambda.$$

Lemma 16 *The following facts hold for any cardinals κ, λ, μ:*

1. $0 \leq \kappa$
2. $\kappa \leq \kappa$
3. $(\kappa \leq \lambda \ \wedge \ \lambda \leq \mu) \ \Rightarrow \ \kappa \leq \mu$
4. $(\kappa \leq \lambda \ \wedge \ \lambda \leq \kappa) \ \Rightarrow \ \kappa = \lambda$
5. $\kappa \leq \lambda \ \vee \ \lambda \leq \kappa$
6. $\kappa < \lambda \ \Leftrightarrow \ (\kappa \leq \lambda \ \wedge \kappa \neq \lambda)$.

Proof. The proofs of all these facts follow easily from the definitions of $<$ and $\leq$, plus known facts and theorems about $\preceq$ and $\prec$. For example, (1) follows from $\varnothing \subseteq X$ for any set X; (2) follows from the fact that id_κ is injective; (3) follows from the fact that the composition of two injective functions is injective (see Exercise 35); (4) is an immediate consequence of the Schroeder-Bernstein Theorem (Theorem 10); (5) follows trivially from *CC* (Corollary 9); and (6) is an easy exercise. □

Note that, because of (4) and (5), given two cardinals κ and λ, we can define $max(\kappa, \lambda) = \textbf{if } \kappa \leq \lambda \textbf{ then } \lambda \textbf{ else } \kappa \textbf{ fi}$, and $min(\kappa, \lambda) = \textbf{if } \kappa \leq \lambda \textbf{ then } \kappa \textbf{ else } \lambda \textbf{ fi}$; and we then have $max(\kappa, \lambda) = max(\lambda, \kappa)$, and $min(\kappa, \lambda) = min(\lambda, \kappa)$.

Note, again, that the interpretation of $<$, $\leq$, *max*, and *min* on $\mathbb{N}$ is *exactly* the strict and nonstrict ordering on the natural numbers, and the *max* and *min* functions. Therefore, following our project of reducing arithmetic to set theory, we could just *define* the order relation on naturals n and m by the defining equivalence

$$n \leq m \ \Leftrightarrow \ n \preceq m.$$

This brings up another way in which the extension from ordinary arithmetic to cardinal arithmetic is so very smooth. It is well-known that addition, multiplication, and exponentiation of numbers are all *monotonic* operations. That is, when we apply them to bigger numbers, we get bigger or equal results (except for a 0 exponent, since $0^0 = 1$, but $0^1 = 0$). But instead of proving such monotonicity results in ordinary arithmetic, which from the vantage point of cardinals is just "baby arithmetic," we can prove them instead for *any* cardinals, thus getting them for $\mathbb{N}$ as a special case.

Theorem 18 *(Monotonicity of Cardinal Arithmetic). For any cardinals $\kappa, \kappa', \lambda, \lambda'$, and I-indexed sets of cardinals $\mu = \{\mu_i\}_{i\in I}$ and $\mu' = \{\mu'_i\}_{i\in I}$, the following implications hold:*

$$\kappa \leq \kappa' \ \wedge\ \lambda \leq \lambda' \quad \Rightarrow \quad \kappa + \lambda \leq \kappa' + \lambda' \tag{15.19}$$

$$\kappa \leq \kappa' \ \wedge\ \lambda \leq \lambda' \quad \Rightarrow \quad \kappa \cdot \lambda \leq \kappa' \cdot \lambda' \tag{15.20}$$

$$\kappa \leq \kappa' \quad \Rightarrow \quad \kappa^\lambda \leq \kappa'^\lambda \tag{15.21}$$

$$\kappa \leq \kappa' \ \wedge\ \lambda \leq \lambda' \ \wedge\ \lambda \neq 0 \quad \Rightarrow \quad \kappa^\lambda \leq \kappa'^{\lambda'} \tag{15.22}$$

$$(\forall i \in I)(\mu_i \leq \mu'_i) \quad \Rightarrow \quad \Sigma_{i\in I}\mu_i \leq \Sigma_{i\in I}\mu'_i \tag{15.23}$$

$$(\forall i \in I)(\mu_i \leq \mu'_i) \quad \Rightarrow \quad \Pi_{i\in I}\mu_i \leq \Pi_{i\in I}\mu'_i \tag{15.24}$$

Proof. The proofs of these implications are all based on the same idea, namely, that, as shown in Exercises 41, 94, and 43, $\oplus$, $\times$, $\bigoplus$ $\bigtimes$, and $[\_\to\_]$ are all *functorial* constructions.

Let us see how the functoriality of $\bigoplus$, $\bigtimes$, and $[\_\to\_]$ respectively imply (12.23), (12.24), (12.21) and (12.22) ((12.19) is a special case of (12.23), and (12.20) is a special case of (12.24)).

In both (12.23) and (12.24), we are assuming that there is an I-indexed injective function $f : \mu \longrightarrow \mu'$. But for each $(x_i, i) \in \bigoplus \mu$ we have $(\bigoplus f)(x_i, i) = (f_i(x_i), i)$. This means that $\bigoplus f$ is injective, since $(\bigoplus f)(x_i, i) = (\bigoplus f)(x'_{i'}, i')$ means that $(f_i(x_i), i) = (f_{i'}(x'_{i'}), i')$, which, by Lemma 1, forces $i = i'$ and $f_i(x_i) = f_i(x'_i)$, which then forces $x_i = x'_i$ by f_i injective, which finally forces $(x_i, i) = (x'_{i'}, i')$. Therefore, we have $\bigoplus \mu \preceq \bigoplus \mu'$, yielding (12.22), as desired.

Similarly, $\bigtimes f$ maps each $\{x_i\}_{i\in I} \in \bigtimes \mu$ to $\{f_i(x_i)\}_{i\in I} \in \bigtimes \mu'$. But this means that $\bigtimes f$ is injective, since $\bigtimes f(\{x_i\}_{i\in I}) = \bigtimes f(\{x'_i\}_{i\in I})$ means that $\{f_i(x_i)\}_{i\in I} = \{f_i(x'_i)\}_{i\in I}$, which then forces $f_i(x_i) = f_i(x'_i)$ for each $i \in I$, which by f_i injective for each $i \in I$ in turn forces $x_i = x'_i$ for each $i \in I$, which means that $\{x_i\}_{i\in I} = \{x'_i\}_{i\in I}$. Therefore, $\bigtimes \mu \leq \bigtimes \mu'$, yielding (12.23), as desired.

Let us prove (12.21). By hypothesis we have an injection $f : \kappa \rightarrowtail \kappa'$. To prove (12.21) we just have to show that $[id_\lambda \to f] : [\lambda \to \kappa] \longrightarrow [\lambda \to \kappa'] : h \mapsto h; f$ is injective. But f is injective and, by Exercise 35, mono. Therefore, $h; f = h'; f$ implies $h = h'$, so that $[id_\lambda \to f]$ is injective, as desired.

Let us finally prove (12.22). By hypothesis we have injections $f : \kappa \rightarrowtail \kappa'$ and $g : \lambda \rightarrowtail \lambda'$, and we know that $\lambda \neq \emptyset$. Also, by (12.1) we know that $\kappa^{\lambda'} \leq \kappa'^{\lambda'}$. Therefore, by the transitivity of $\leq$, all we need to prove is $\kappa^\lambda \leq \kappa^{\lambda'}$. But by $\lambda \neq \emptyset$ and Exercise 34, we know that $g : \lambda \rightarrowtail \lambda'$ has a right inverse, say, $h : \lambda' \twoheadrightarrow \lambda$, with $g; h = id_\lambda$. If we can show that $[h \to id_\kappa] : [\lambda \to \kappa] \longrightarrow [\lambda' \to \kappa] f \mapsto h; f$ is injective, we will be done, since this will give us $\kappa^\lambda \leq \kappa^{\lambda'}$. But by Exercise 34, h is surjective, and by Exercise 35, h is then epi. Therefore, $h; f = h; f'$ implies $f = f'$, so that $[h \to id_\kappa]$ is injective, as desired. □

Exercise 106 *Prove that for κ and λ any two cardinals such that $\kappa, \lambda \geq 2$, we always have $\kappa + \lambda \leq \kappa \cdot \lambda$.*

The statement and proof of several other basic results about cardinal arithmetic is postponed until §18.5.2. However, in what follows we prove some cardinality results for $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$.

15.2 The Integers and the Rationals are Countable

It is trivial to show that $\mathbb{N} \cong \mathbb{Z}$, and therefore that $|\mathbb{N}| = |\mathbb{Z}|$. We can, for example, use the bijective function $\mathbb{N} \longrightarrow \mathbb{Z} : n \mapsto$ **if** $even(n)$ **then** $n/2$ **else** $-(s(n)/2)$ **fi**. Almost the same argument shows that $|\mathbb{N}| = |\mathbb{N} \oplus \mathbb{N}| = |\mathbb{N}| + |\mathbb{N}|$, since the function $\mathbb{N} \longrightarrow \mathbb{N} \oplus \mathbb{N} : n \mapsto$ **if** $even(n)$ **then** $(n/2, 0)$ **else** $((n-1)/2, 1)$ **fi** is bijective. In fact, we already knew that this bijection existed, since we saw in §6.5 that we had an abstract disjoint union decomposition $\mathbb{N} \oplus \mathbb{N} = \mathbb{N}$ using even and odd numbers and, by Exercise 40, we know that all abstract disjoint unions are isomorphic and therefore in bijection with each other. We also essentially *know* already that $|\mathbb{N}| = |\mathbb{N} \times \mathbb{N}| = |\mathbb{N}| \cdot |\mathbb{N}|$, since by (10.4) and (10.20) we know that $|\mathbb{N}| = |\mathbb{N}| \cdot 1 \leq |\mathbb{N} \times \mathbb{N}| = |\mathbb{N}| \cdot |\mathbb{N}|$, and we already know that $|\mathbb{N}| \cdot |\mathbb{N}| = |\mathbb{N} \times \mathbb{N}| \leq |\mathbb{N}|$, since we saw in §6.5 that the subset $\{2^{n+1} \cdot 3^{m+1} \in \mathbb{N} \mid n, m \in \mathbb{N}\} \subset \mathbb{N}$ is such that $\{2^{n+1} \cdot 3^{m+1} \in \mathbb{N} \mid n, m \in \mathbb{N}\} = \mathbb{N} \times \mathbb{N}$ as an abstract product and, again by Exercise 40, we know that all abstract products are isomorphic and therefore in bijection with each other.

Exercise 107 *Prove the following:*

1. $(\forall n \in \mathbb{N})\ |\mathbb{N}| + n = |\mathbb{N}|$.

2. *For any infinite cardinal* κ, $(\forall n \in \mathbb{N})\ \kappa + n = \kappa$.

3. *For any infinite cardinal* κ, $\kappa + |\mathbb{N}| = \kappa$.

Since the natural numbers are (up to the change of representation $n \mapsto n/1$) a subset of the rationals, we have $\mathbb{N} \preceq \mathbb{Q}$. However, $\mathbb{Q}$, seems much bigger than $\mathbb{N}$. After all, $\mathbb{Q}$ is a so-called "dense set" inside the set $\mathbb{R}$ of real numbers, so that every real number is a limit point of a sequence of rational numbers. For example, π is the limit point of the sequence of rational numbers

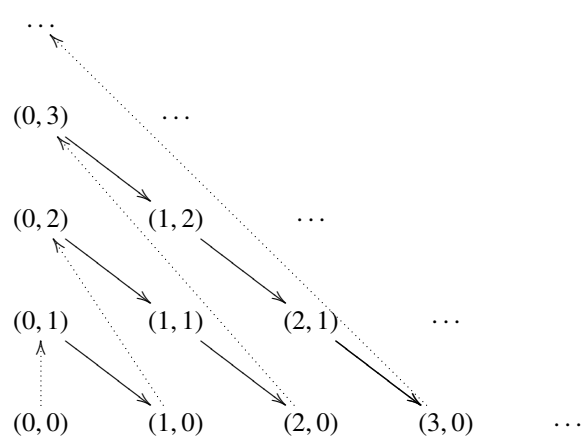
$$3,\quad 3.1,\quad 3.14,\quad 3.141,\quad 3.1415,\quad 3.14159,\quad \ldots$$

And there are good reasons to believe (in fact we will prove this in §15.3) that $\mathbb{N} \prec \mathbb{R}$, so that $\mathbb{R}$ has a strictly greater form of infinity than $\mathbb{N}$. Therefore, it is somewhat surprising (at least for a newcomer to set theory) that $\mathbb{N} \cong \mathbb{Q}$, and therefore that $|\mathbb{N}| = |\mathbb{Q}|$. This result was proved in a beautifully simple way by Georg Cantor, the pioneering creator of set theory.

Theorem 19 $|\mathbb{N}| = |\mathbb{Q}|$.

Proof. Since $\mathbb{Q}$ is constructed as a quotient set of $\mathbb{Z} \times (\mathbb{Z} - \{0\})$ by an equivalence relation, we have a surjective function $\mathbb{Z} \times (\mathbb{Z} - \{0\}) \twoheadrightarrow \mathbb{Q}$, and therefore, by Theorem 11, $\mathbb{Q} \preceq \mathbb{Z} \times (\mathbb{Z} - \{0\})$. But $|\mathbb{Z} \times (\mathbb{Z} - \{0\})| = |\mathbb{Z}| \cdot |(\mathbb{Z} - \{0\})| = |\mathbb{N}| \cdot |\mathbb{N}|$. But then the equality $|\mathbb{N}| \cdot |\mathbb{N}| = |\mathbb{N}|$, which we already know because $\{2^{n+1} \cdot 3^{m+1} \in \mathbb{N} \mid n, m \in \mathbb{N}\} = \mathbb{N} \times \mathbb{N}$, forces $|\mathbb{Q}| \leq |\mathbb{N}|$ and $|\mathbb{N}| \leq |\mathbb{Q}|$, so that $|\mathbb{N}| = |\mathbb{Q}|$, as desired. □

Cantor's proof that $|\mathbb{N}| \cdot |\mathbb{N}| = |\mathbb{N}|$ was different, and worth recalling because of its beauty and simplicity. He proved $|\mathbb{N}| \cdot |\mathbb{N}| = |\mathbb{N}|$ by observing that there is a bijective function $f : \mathbb{N} \longrightarrow \mathbb{N} \times \mathbb{N}$ that enumerates the set $\mathbb{N} \times \mathbb{N}$ according to the method of successively enumerating the increasingly bigger diagonals in the picture:



Note that f^{-1} is the function $f^{-1} : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N} : (n, m) \mapsto (\Sigma_{1 \leq i \leq (n+m)} i) + n$.

Corollary 4 *(Countable Unions of Countable Sets are Countable). Let* $A = \{A_n\}_{n \in \mathbb{N}}$ *be an* $\mathbb{N}$*-indexed set such that for each* $n \in \mathbb{N}$, $|A_n| = |\mathbb{N}|$. *Then* $|\bigcup A| = |\mathbb{N}|$.

Proof. Obviously, since for each A_n we have $A_n \subset \bigcup A$, and $|A_n| = |\mathbb{N}|$, we then have $|\bigcup A| \geq |\mathbb{N}|$. So we just need to show $|\bigcup A| \leq |\mathbb{N}|$. But note that: (i) the projection function $p_1 : \bigoplus A \longrightarrow \bigcup A : (a_n, n) \mapsto a_n$ is surjective, (ii) since $|A_n| = |\mathbb{N}|$ for each $n \in \mathbb{N}$, we have $|\bigoplus A| = |\bigoplus \mathbb{N}_{\mathbb{N}}|$; and (iii) by Exercise 90 we have $\bigoplus \mathbb{N}_{\mathbb{N}} = \mathbb{N} \times \mathbb{N}$. Therefore, we have

$$|\bigcup A| \leq |\bigoplus A| = |\mathbb{N}| \cdot |\mathbb{N}| = |\mathbb{N}|$$

as desired. □

15.3 The Continuum and the Continuum Hypothesis

The real numbers $\mathbb{R}$ are sometimes called the *continuum* for obvious geometrical reasons: when we interpret $\mathbb{R}$ geometrically as the so-called real line, it is a continuous, uninterrupted line that has no "holes." Instead, the rational line $\mathbb{Q} \subset \mathbb{R}$, although dense in $\mathbb{R}$ (it fills all of $\mathbb{R}$ if we add to it all its limit points), *does* have holes such as, for example, the points corresponding to the "irrational" numbers $\sqrt{2}$, π, e, and so on. Intuitively, therefore, the continuum should have a strictly greater kind of infinity than the countable infinity of the natural numbers. That is, we should have $|\mathbb{N}| < |\mathbb{R}|$. We can therefore ask two questions. First, is this intuition correct? And, second, can we *characterize* the exact size of $\mathbb{R}$ in simpler terms? Georg Cantor answered the second question by showing that $|\mathbb{R}| = |\mathcal{P}(\mathbb{N})|$, thus giving also an affirmative answer to the first question, since by Cantor's Theorem (Theorem 9) we have $|\mathbb{N}| < |\mathcal{P}(\mathbb{N})|$. The proof below is essentially Cantor's original proof (see [9] §4, [488]).

Theorem 20 *(Cantor).* $|\mathbb{R}| = |\mathcal{P}(\mathbb{N})|$.

Proof. First of all, note that the cotangent function $cot = \lambda x.\ cos(x)/sin(x)$ gives us a *bijection* $cot : (0, \pi) \longrightarrow \mathbb{R}$, and therefore we also have a bijection $\lambda x.\ cot(\pi \cdot x) : (0, 1) \longrightarrow \mathbb{R}$. Therefore, it is enough to prove $|(0, 1)| = |\mathcal{P}(\mathbb{N})|$, or, equivalently, $|(0, 1)| = |2^{\mathbb{N}}|$. Next note that, by expressing each number $x \in (0, 1)$ in its binary expansion $x = 0.b_1 b_2 \ldots b_n \ldots$, with $b_{s(i)} \in \{0, 1\}$ for each $i \in \mathbb{N}$, we can define an *injective* function $j : (0, 1) \rightarrowtail 2^{\mathbb{N}}$, where j maps each $x = 0.b_1 b_2 \ldots b_n \ldots$ to the function $\lambda n \in \mathbb{N}.\ b_{s(n)}$. We only have to take care of defining j properly for the cases where $x = k/2^n$, with $1 \leq k \leq 2^n$, and $n \geq 1$, since those numbers are the only numbers in $(0, 1)$ that have *two* different binary expansions, namely, if we can write the number k as a sum $k = b_1 \cdot 2^{n-1} + b_2 \cdot 2^{n-2} + \ldots + b_{n-1} \cdot 2 + b_n$, with $b_n = 1$ and $b_i \in \{0, 1\}$, $1 \leq i \in n$, then $x = k/2^n$ has the binary expansion $k/2^n = 0.b_1 b_2 \ldots b_{n-1} 10000000 \ldots$, and also the binary expansion $k/2^n = 0.b_1 b_2 \ldots b_{n-1} 01111111 \ldots$. So, to define j properly we map each $x = k/2^n$, with $k = b_1 \cdot 2^{n-1} + b_2 \cdot 2^{n-2} + \ldots + b_{n-1} \cdot 2 + b_n$ and $b_n = 1$, to the function

$$j(k/2^n) = \lambda x \in \mathbb{N}.\ \textbf{if}\ s(x) < n\ \textbf{then}\ b_{s(x)}\ \textbf{else if}\ s(x) = n\ \textbf{then}\ 0\ \textbf{else}\ 1\ \textbf{fi fi}.$$

The key observation, now, is that the set $2^{\mathbb{N}} - j[(0, 1)]$ consists exactly of: (i) the constant function $\lambda x \in \mathbb{N}.\ 1$, corresponding to 1 in the binary expansion $0.1111111 \ldots$, and (ii) those functions $f : \mathbb{N} \longrightarrow 2$ such that there is an n such that $(\forall m \in \mathbb{N})\ (m \geq n) \Rightarrow f(m) = 0$, which correspond to either the infinite expansion of 0 as $0 = 0.0000000 \ldots$, or to the alternative infinite expansion $k/2^n = 0.b_1 b_2 \ldots b_{n-1} 10000000 \ldots$ for some $k/2^n$. But note that the set of functions of type (ii) can be mapped bijectively to $\mathbb{N}$, namely, the function constantly 0 can be mapped to 0, and the function corresponding to the binary expansion $0.b_1 b_2 \ldots b_{n-1} 10000000 \ldots$ can be mapped to the number $1 b_{n-1} \ldots b_2 b_1$ in binary notation. Therefore we have $|2^{\mathbb{N}} - j[(0, 1)]| = 1 + |\mathbb{N}| = |\mathbb{N}|$. As a consequence, using Exercise 107-(3), we have:

$$|\mathbb{R}| = |(0, 1)| = |j[(0, 1)]| = |j[(0, 1)]| + |\mathbb{N}| = |j[(0, 1)]| + |2^{\mathbb{N}} - j[(0, 1)]| = |2^{\mathbb{N}}|,$$

as desired. □

The following lemma gives us very useful information about another cardinal we had not yet considered, namely, $|\mathbb{N}|^{|\mathbb{N}|}$.

Lemma 17 $|\mathbb{N}|^{|\mathbb{N}|} = 2^{|\mathbb{N}|}$.

Proof. Since $2 \subset \mathbb{N}$, we have $[\mathbb{N} \to 2] \subset [\mathbb{N} \to \mathbb{N}]$, and therefore $|\mathbb{N}|^{|\mathbb{N}|} \geq 2^{|\mathbb{N}|}$. We show $|\mathbb{N}|^{|\mathbb{N}|} \leq 2^{|\mathbb{N}|}$ by means of the function $f : [\mathbb{N} \to \mathbb{N}] \rightarrowtail [\mathbb{N} \to 2]$, which maps each $\{k_n\}_{n \in \mathbb{N}} \in [\mathbb{N} \to \mathbb{N}]$ to the sequence

$$1, \overset{k_0+1}{\ldots} 1,\ 0,\ 1, \overset{k_1+1}{\ldots} 1,\ 0,\ \ldots\ 1, \overset{k_n+1}{\ldots} 1,\ 0\ \ldots$$

which is clearly injective, since if $\{k_n\}_{n \in \mathbb{N}} \neq \{j_n\}_{n \in \mathbb{N}}$ have i as the first position in which they differ, then the i-th block of 1's will be the first moment in which $f(\{k_n\}_{n \in \mathbb{N}})$ and $f(\{j_n\}_{n \in \mathbb{N}})$ differ. □

Since the proof of Lemma 17 is independent from the proof of Theorem 20, Lemma 17 gives us an alternative path to prove Theorem 20. Specifically, we can prove $|\mathbb{R}| = |\mathcal{P}(\mathbb{N})|$ by using the representation of a real number as an equivalence class of Cauchy sequences. Since Cauchy sequences are a subset of $[\mathbb{N} \to \mathbb{Q}]$, and $[\mathbb{N} \to \mathbb{Q}] \cong [\mathbb{N} \to \mathbb{N}]$, we get $|\mathbb{R}| \leq |\mathbb{N}|^{|\mathbb{N}|}$, so we only have to prove $|\mathbb{R}| \geq |\mathcal{P}(\mathbb{N})|$.

Lemma 17 has also the following useful corollary:

Corollary 5 *For each $n \in \mathbb{N}$ such that $n \geq 2$ we have $n^{|\mathbb{N}|} = 2^{|\mathbb{N}|}$.*

Proof. Since $2 \subseteq n \subset \mathbb{N}$, we have $2^{|\mathbb{N}|} \leq |n|^{|\mathbb{N}|} \leq |\mathbb{N}|^{|\mathbb{N}|}$. But since $|\mathbb{N}|^{|\mathbb{N}|} = 2^{|\mathbb{N}|}$, we get $|n|^{|\mathbb{N}|} = 2^{|\mathbb{N}|}$, as desired. □

Another useful result, equivalent to $|\mathbb{R}| + |\mathbb{R}| = |\mathbb{R}|$, is the following lemma:

Lemma 18 $2^{|\mathbb{N}|} + 2^{|\mathbb{N}|} = 2^{|\mathbb{N}|}$.

Proof. Since we have an injection $i_1 : [\mathbb{N}{\to}2] \rightarrowtail [\mathbb{N}{\to}2] \oplus [\mathbb{N}{\to}2]$, we trivially have $2^{|\mathbb{N}|} + 2^{|\mathbb{N}|} \geq 2^{|\mathbb{N}|}$. To see $2^{|\mathbb{N}|} + 2^{|\mathbb{N}|} \leq 2^{|\mathbb{N}|}$ we can use the clearly injective function $f : [\mathbb{N}{\to}2] \oplus [\mathbb{N}{\to}2] \rightarrowtail [\mathbb{N}{\to}2]$ that maps $(\{b_n\}, 0)$ to the sequence

$$1,\ 0,\ b_0,\ b_1,\ \ldots b_n,\ \ldots$$

and $(\{b_n\}, 1)$ to the sequence

$$1,\ 1,\ 0,\ b_0,\ b_1,\ \ldots b_n,\ \ldots$$

Yet another useful result is the following:

Lemma 19 $2^{|\mathbb{N}|} \cdot 2^{|\mathbb{N}|} = 2^{|\mathbb{N}|}$.

Proof. It is enough to show $[\mathbb{N}{\to}2] \times [\mathbb{N}{\to}2] \cong [\mathbb{N}{\to}2]$. This is shown by the function *merge* : $[\mathbb{N}{\to}2] \times [\mathbb{N}{\to}2] \longrightarrow [\mathbb{N}{\to}2] : (a, b) \mapsto \lambda n \in \mathbb{N}.$ **if** $even(n)$ **then** $a(n/2)$ **else** $b((n-1)/2)$ **fi** $\in 2$. *merge* is bijective, because its inverse is the function *split* : $[\mathbb{N}{\to}2] \longrightarrow [\mathbb{N}{\to}2] \times [\mathbb{N}{\to}2] : a \mapsto (double; a, double + 1; a)$, where $double = \lambda x \in \mathbb{N}.\ (2 \cdot x) \in \mathbb{N}$, and $double + 1 = \lambda x \in \mathbb{N}.\ ((2 \cdot x) + 1) \in \mathbb{N}$. □

Exercise 108 *Give a completely different proof of Lemma 19 by using instead Exercise 90 and Cantor's bijection $\mathbb{N} \cong \mathbb{N} \times \mathbb{N}$.*

15.3.1 Peano Curves

Since by Theorem 20 we have $|\mathbb{R}| = |\mathcal{P}(\mathbb{N})| = |[\mathbb{N}{\to}2]| = 2^{|\mathbb{N}|}$, the above lemma shows that $|\mathbb{R}| \cdot |\mathbb{R}| = |\mathbb{R}|$, and therefore, that there is a bijection $\mathbb{R} \cong \mathbb{R}^2$. This was already shown by Cantor towards the end of the 19th century and caused a big stir, since it showed that the real line and the real plane can be put in bijective correspondence with each other and therefore have the same size! Encouraged by this result of Cantor's, Giuseppe Peano proved an equally amazing result: he gave a detailed construction of a *continuous* and *surjective* function *peano* : $\mathbb{R} \twoheadrightarrow \mathbb{R}^2$, that is, of a continuous *curve* that fills in the entire 2-dimensional plane! This was considered impossible at the time, since how can a 1-dimensional object such as a continuous curve (which we can imagine as an infinitely thin string meandering around the plane) fill in the entire plane, which is a 2-dimensional object?

The reason why this astonished mathematicians at the time was that the curves they knew about were all differentiable (or piecewise differentiable) curves, and indeed such curves can never fill in the plane. Peano's curve —and similar curves proposed later, such as one by David Hilbert— was a fractal-like curve, constructed as a limit of curves, which is not differentiable at all. Successive approximations for both the Peano and the Hilbert curve are shown in Figure 15.3.1.

But why stopping at the plane? The same idea can be used to construct a continuous curve that fills in any $n+1$-dimensional Euclidean space! We just compose the following sequence of surjective continuous functions:

$$\mathbb{R} \overset{peano}{\twoheadrightarrow} \mathbb{R}^2 \overset{id_{\mathbb{R}} \times peano}{\twoheadrightarrow} \mathbb{R}^3 \quad \ldots \quad \mathbb{R}^n \overset{id_{\mathbb{R}^{n-1}} \times peano}{\twoheadrightarrow} \mathbb{R}^{n+1}$$

15.3.2 The Continuum Hypothesis

Since we know that $|\mathbb{R}| = |\mathcal{P}(\mathbb{N})|$, and by Cantor's Theorem $|\mathbb{N}| < |\mathcal{P}(\mathbb{N})|$, we know that the continuum has a strictly bigger kind of infinity than the natural numbers. Furthermore, by Theorem 13, we know that countably infinite sets are the *smallest* possible infinite sets, since for any infinite set A, $\mathbb{N} \preceq A$. But which is the *smallest uncountable* infinite cardinal? Is $|\mathcal{P}(\mathbb{N})|$ the smallest such cardinal?

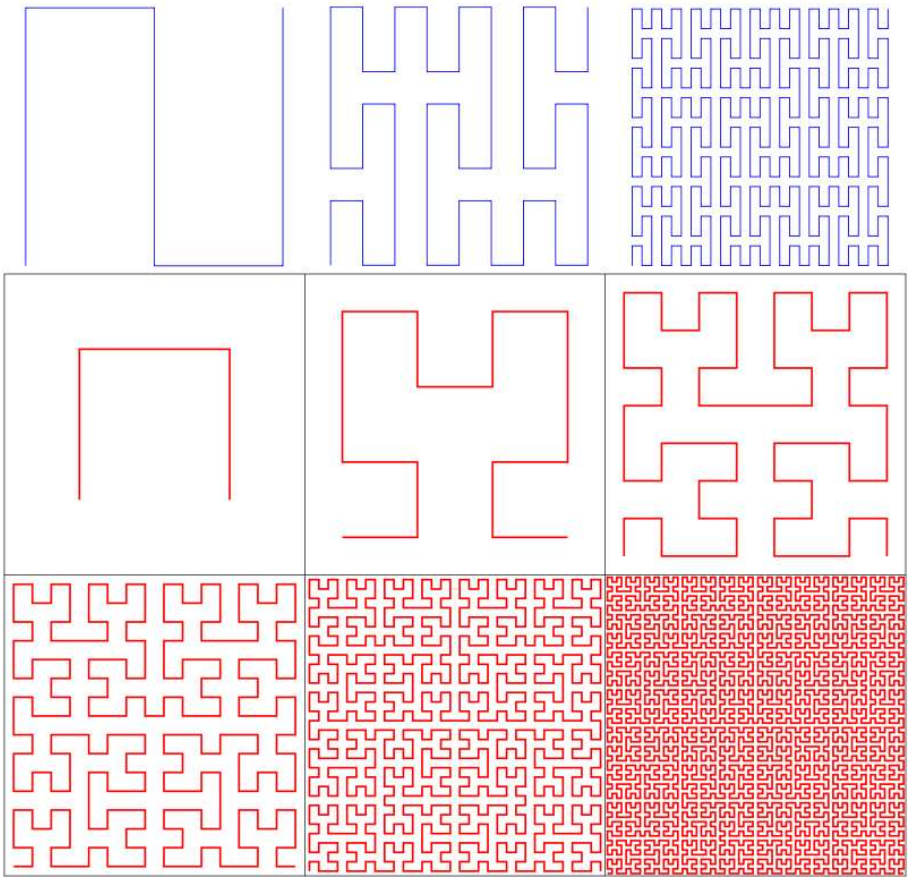


Figure 15.1: Three successive approximations for a Peano curve filling the unit square are shown in the first row. The second and third rows show six successive approximations for a Hilbert curve. The pictures, with a high-level introduction, appear in: `http://en.wikipedia.org/wiki/Space-filling-curve`.

By cardinal comparability (*CC*), given any two infinite uncountable sets A and B, either $|A| \leq |B|$, or $|B| \leq |A|$. So the question becomes: is there an uncountable infinite set A such that $|A| < |\mathcal{P}(\mathbb{N})|$, or, equivalently, is there an uncountable infinite subset $A \subset \mathbb{R}$ of the real numbers such that $|A| < |\mathbb{R}|$?

Georg Cantor tried for many years to prove that the answer is *no*: that there is no other order of infinity between countable infinity and the continuum, because the continuum has the smallest possible type of uncountable infinity. But he could not obtain a proof. Therefore, Cantor's conjecture became a hypothesis, the so-called *Continuum Hypothesis* (*CH*). Settling whether *CH* is true or not was Problem 1 in David Hilbert's famous list of open problems presented the year 1,900 in Paris at the Second International Congress of Mathematics. *CH* can be formalized precisely as the following formula in set theory:

$$(CH) \quad (\forall Y \subseteq \mathbb{R})(|Y| \leq |\mathbb{N}| \ \vee \ |Y| = |\mathbb{R}|).$$

But what about $\mathcal{P}(\mathbb{R})$? Is there some distinct order of infinity in between that of the continuum and that of $\mathcal{P}(\mathbb{R})$, which, again by Cantor's Theorem, we know to be strictly bigger than that of the continuum? If we guessed that the answer to the previous question was *no*, we would tend to favor a no answer also in this case. The point is that we can extend the continuum hypothesis to all infinite sets by means of the, so-called *Generalized Continuum Hypothesis* (*GCH*), which can be formalized in set theory as the formula:

$$(GCH) \quad (\forall X)((|X| \geq |\mathbb{N}|) \ \Rightarrow \ ((\forall Y \subseteq \mathcal{P}(X))(|Y| \leq |X| \ \vee \ |Y| = |\mathcal{P}(X)|))).$$

In the late 1930's Kurt Gödel proved (published in fuller form in [20]), that both *AC* and *GCH* are *consistent* with the other axioms (*ZF*) of set theory; that is, that adding either the axiom *AC* to *ZF* (thus getting *ZFC*), or *GCH* to *ZFC*, or adding both *AC* and *GCH*, does not lead to a contradiction.[1] However, Gödel found *GCH* implausible for various reasons and anticipated that it would be *independent* of *ZFC* (that is, that both (*ZFC* + *GCH*) and (*ZFC* + $\neg$*GCH*) would be consistent theories (see Godel's own remarks in his excellent and very readable paper [21], and Feferman's historical remarks about Gödel on this matter in [19], pp. 151–153). The independence of *GCH* had also been anticipated by Skolem as early as 1920 (see [52], 290–301). In the 1960's Paul Cohen proved that it is also consistent with the *ZFC* axioms of set theory to assume that *GCH* does *not* hold, thus proving the independence of *GCH* and settling that within *ZFC* there is no fact of the matter about this axiom: one can develop set theory with or without *GCH* (see [11] and [30] for detailed expositions of Cohen's results). In [21], Gödel viewed the expected independence of *GCH* from *ZFC* as evidence, not of the impossibility of ever settling *GCH*, but of the need for new set theory axioms stronger than those of *ZFC*. Stronger axioms for which clear justifications can be given and capable of settling *GCH* have been slow in coming; but see [54] for some recent advances on attacking Hilbert's first problem.

CH and *GCH* have a bearing on the interpretation of the so-called *aleph* notation for infinite cardinals, where instances of $\aleph$ (the first letter of the Hebrew alphabet), subindexed by certain "ordinals," are used to denote increasingly bigger infinite cardinals (the $\aleph$ sequence is defined in detail in §18.5.1). Note that if cardinals were to form a set, then by Lemma 16 they would form a *chain* under the $\leq$ order. Since they do not form a set, we can intuitively think of cardinals as linearly arranged in a "mega-chain," too big to be a set (cardinals form a *class*, as explained in §16 and in §18.5.1). Adopting *GCH* implies that the infinite cardinal portion of this "mega-chain" begins as the increasing sequence

$$\aleph_0 < \aleph_1 < \aleph_2 < \ldots < \aleph_n < \ldots$$

where, since for any infinite set A we have $|\mathbb{N}| \leq |A|$, we must have $\aleph_0 = |\mathbb{N}|$. *CH* implies that $\aleph_1 = |\mathcal{P}(\mathbb{N})| = |\mathbb{R}|$, and *GCH* further postulates that $\aleph_2 = |\mathcal{P}^2(\mathbb{N})|, \ldots, \aleph_n = |\mathcal{P}^n(\mathbb{N})|$, and so on.

[1] For a brief summary of Gödel's notion of *constructible set* used in these consistency results see Footnote 4 in §18.4.3.

Chapter 16

Classes, Intensional Relations and Functions, and Replacement

What is the status of totalities such as the totality of all cardinals that are too big to be a set? Similarly, what is the status of relations such as $\subseteq$ and functions such as $\lambda(A, B).\ A \cup B$ that make sense for any two sets in general? Althoug there are no sets that we can associate to these notions, they can nevertheless be precisely described in the *formal language* of set theory. They also allow us to obtain a more powerful, yet safe, variant of the comprehension scheme called the *axiom of replacement*.

16.1 Classes

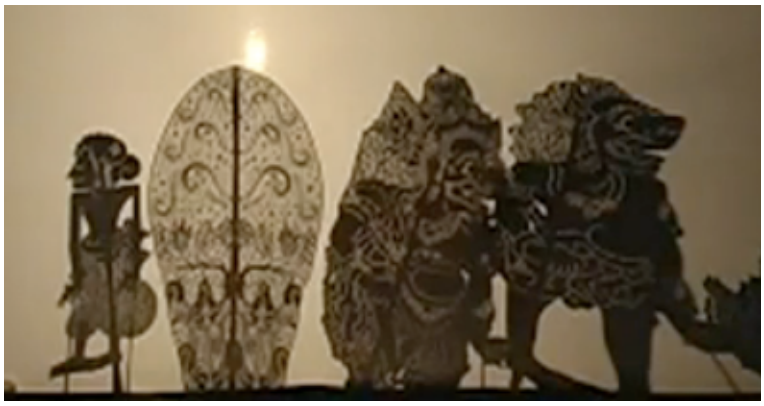
Russell showed that Frege's unrestricted use of the Comprehension Axiom leads to inconsistent notions such as "the set of all noncircular sets" or "the set of all sets" and to vicious contradictions. Full comprehension and such contradictory notions had to be abandoned in order for set theory to be consistent. However, there is an obvious, intuitive sense in which such *totalities* are perfectly reasonable notions, not involving any contradiction. After all, whenever we use the universal quantifier $(\forall x)$ in set theory, we are quantifying over *all sets*, that is, the variable x implicitly ranges over sets in the "universe" $\mathscr{U}$ of all sets. Similarly, all the set-theoretic operations we have defined in §4 are constructions whose arguments range over sets in such a universe $\mathscr{U}$ of all sets. For example, the (binary) set union, resp., cartesian product, operations take any two sets A and B in the universe $\mathscr{U}$ and return a new set $A \cup B$, resp., $A \times B$, also in $\mathscr{U}$.

Likewise, the concept of the totality NC of noncircular sets is not at all contradictory. In fact, we shall see in §19.3 that the axiom (*Found*) of Foundation implies that the totalities $\mathscr{U}$ and NC *coincide*. Furthermore, we can use our set theory language to refer to totalities such as NC and to make assertions about them. For example, the claimed coincidence between $\mathscr{U}$ and NC can be expressed by the formula $(\forall x)\ x \notin x$, which can be shown to be a theorem of set theory using the (*Found*) axiom.

Therefore, what *is* contradictory is not the existence of totalities such as $\mathscr{U}$ and NC. The contradiction comes *only* from assuming that such totalities *are always sets*, and that therefore the membership relation can be applied to them as well. This leads to unsurmountable paradoxes, because assuming that $\mathscr{U}$ and NC are sets we can then form nonsensical formulas such as $\mathscr{U} \in \mathscr{U}$ or $NC \in NC$. There is, however, a straightforward way to rescue Frege's intuition that a set theory formula defines a totality from the clutches of contradiction, namely, to point out that, indeed, any set theory formula φ with a single free variable x defines the totality of all sets that satisfy the property φ; but that in general such a totality is *not* a set, although it may happen to define a set in some particular cases. For example the totality $\mathscr{U}$ can be defined by the formula $x = x$, and the totality NC by the formula $x \notin x$, but they are not sets. Nevertheless, in some cases totalities defined by a formula do define sets; for example, the totality defined by the formula $x \in \varnothing$ does define the empty set $\varnothing$; and the totality defined by the formula $x = \varnothing$ does define the singleton set $\{\varnothing\}$.

Totalities defined by properties φ ranging over sets are called *classes*. Therefore, the universe $\mathscr{U}$ of all sets is a class but (as shown in Theorem 1) *not* a set. Since in Zermelo-Fraenkel set theory variables only range over sets, in *ZFC* we can only talk about classes *indirectly*, namely, through the logical properties φ

that define them. Also, our way of referring to classes in *ZFC* is *intensional*,[1] that is, through a piece of *language* such as the formula φ used to define the class, and not *extensional*, that is, not through an entity in our model corresponding to such a totality, since the only extensional entities in *ZFC* are sets. An analogy with *Balinese shadow theater* may help to illustrate the indirect and intensional status of classes in *ZFC*. Bali's folklore includes puppet theater performances (similar to a Western Guignol) in which shilouette-like puppets are operated behind a cloth screen illuminated by a light source, so that their (possibly much bigger) shadow is then cast on the screen watched by the audience. In this analogy, a set theory formula φ plays the role of a Balinese puppet. Both φ and the puppet are small, *indirect* devices, used to obtain the (much bigger) direct effects of the totality we want to describe and the shadow cast on the screen. And both are in a sense *intensional*, that is, devices used to obtain actual *extensions*, namely, the actual totality of sets characterized by φ, and the actual shadow cast by the puppet.



From Balinese Shadow Theatre: http://www.youtube.com/watch?v=WuN6bIuUK1g

All this means that in *ZFC* classes are in some sense "second-class citizens," not enjoying the first-class status of sets, which are the only entities directly available in the intended model. But this is just a side effect of how the particular first-order language of *ZFC* is set up, and can be easily corrected, so that both classes and sets enjoy first-class status. This is precisely what the alternative axiomatic set theory of von Neumann-Gödel-Bernays (*vNGB*) accomplishes. In the language of *vNGB* both sets and classes are first-class citizens, and we have two different sets of variables: lower-case variables $x, y, z, \ldots$ ranging over sets, and upper-case variables $X, Y, Z, \ldots$ ranging over classes. Both *ZFC* and *vNGB* have their own technical advantages and disadvantages. On the one hand, *ZFC* has a simpler, more minimal language; but on the other, *vNGB*, besides supportting classes explicitly, has the important advantage of having a *finite* set of axioms.[2] Instead, *ZFC* has a finite set of axiom *schemes*, but some of its axioms (namely (*Sep*) and (*Rep*)) by being schematic correspond not to a single axiom, but to a *countable family* of such axioms. Nevertheless, the differences between *ZFC* and *vNGB* are *merely technical*, since they can prove exactly the same theorems. More precisely, if φ is a formula in the language of *ZFC*, then we have the equivalence

$$ZFC \vdash \varphi \quad \Leftrightarrow \quad vNGB \vdash \varphi.$$

For a detailed description of the language and axioms of *vNGB* and a proof of the above equivalence of proving power between *ZFC* and *vNGB* see [11].

Definition 25 *A* class *is a formula φ in the language of ZFC having a single free variable x. A class is called* set-defining *iff there is a set A such that the equivalence $\varphi \Leftrightarrow x \in A$ is a theorem of set theory. A class is called* proper *iff it is not set-defining.*

For example, the classes $x \in \emptyset$ and $x = \emptyset$ are set-defining, with respective sets $\emptyset$ and $\{\emptyset\}$, whereas $x = x$ and $x \notin x$ are proper classes. We will use capitalized script letters $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$, and so on, to denote classes.

[1] The intensional vs. extensional distinction will be further discussed later in this chapter; for the moment it is enough to think of "intensional" as "language-based" and of "extensional" as "set-based" or, more generally, "model-based."

[2] Essentially, because instead of having to use set theory formulas in the separation and replacement axiom schemes, one can use class variables and a single axiom instead of a scheme.

That is, we will feel free to extend the language of set theory with additional unary predicate symbols such as $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$, etc., to define specific classes. Furthermore, we will use the more suggestive postfix notation $x : \mathscr{A}$ instead than the usual prefix notation $\mathscr{A}(x)$ to indicate that $\mathscr{A}$ holds on x. Intuitively, we read $x : \mathscr{A}$ as "the set x *belongs* to the class $\mathscr{A}$." In this way we avoid the notation $x \in \mathscr{A}$, which would be incorrect in *ZFC*, since in *ZFC* the predicate $\_ \in \_$ should only hold between two sets. If the class named $\mathscr{A}$ is defined by the formula φ, this means that we have extended the language of set theory by means of the defining equivalence

$$x : \mathscr{A} \quad \Leftrightarrow \quad \varphi.$$

For example, we have already introduced the notation $\mathscr{U}$ for the class defining the entire universe of sets, which means that we are defining the unary predicate $\mathscr{U}$ by means of the equivalence

$$x : \mathscr{U} \quad \Leftrightarrow \quad x = x.$$

Let us give a few more examples. The following classes are all proper.

- The class **Nonmt** of *nonempty sets*, defined by the formula $x \neq \emptyset$.
- The class $\mathscr{U}_{fin}$ of *finite sets*, defined by the formula $(\exists n \in \mathbb{N})\ x \cong n$.
- The class **Sing** of *singleton sets*, defined by the formula $(\exists y)\ x = \{y\}$.
- The class $\mathscr{U}^2$ of *ordered pairs*, defined by the formula $(\exists x_1)(\exists x_2)\ x = (x_1, x_2)$.
- More generally, the class $\mathscr{U}^n$ of *n-tuples*, $(n \geq 2)$, defined by the formula $(\exists x_1)(\exists x_2)\ldots(\exists x_n)\ x = (x_1, x_2, \ldots, x_n)$.
- The class **Rel** of relations, defined by the formula $(\exists y, z)\ x \subseteq y \times z$.
- The class **Fun** of functions, defined by the formula $(\exists y, z)\ x \in [y \to z]$.
- The class **Graph** of graphs, defined by the formula $(\exists N, G)\ x = (N, G) \wedge G \subseteq N \times N$.
- The class **Poset** of posets (with strict order), defined by the formula $(\exists A, R)\ x = (A, R) \wedge R \subseteq A \times A \wedge (\forall y \in A)\,(y, y) \notin R \wedge (\forall y, z, w \in A)\,((y, z) \in R \wedge (z, w) \in R) \Rightarrow (y, w) \in R$.
- The class **WFRel** of well-founded relations, defined by the formula $(\exists A, R)\ x = (A, R) \wedge R \subseteq A \times A \wedge (\forall B \in (\mathcal{P}(A) - \{\emptyset\}))\,(\exists b \in B)\,((\forall y \in A)\,(b, y) \in R \Rightarrow y \notin B)$.
- The class **Group** of groups, defined by the formula $(\exists G, e, i, m)\ x = (G, e, i, m) \wedge e \in G \wedge i \in [G \to G] \wedge m \in [G \times G \to G] \wedge (\forall y \in G)\ m(y, e) = y \wedge (\forall y \in G)\ m(y, i(y)) = e \wedge (\forall y, z, w \in G)\ m(m(y, z), w) = m(y, m(z, w))$.
- The class **Peano-Lawvere** of structures satisfying the properties stated in the Peano-Lawvere axiom[3] for the natural numbers, defined by the formula $(\exists \mathbb{N}, o, s)\ x = (\mathbb{N}, 0, s) \wedge 0 \in \mathbb{N} \wedge s \in [\mathbb{N} \to \mathbb{N}] \wedge (\forall A, a, f)\ (a \in A \wedge f \in [A \to A]) \Rightarrow (\exists! g \in [\mathbb{N} \to A])\ g(0) = a \wedge g; s = f; g$.
- The class **Top** of topological spaces, defined by the formula $(\exists A, \mathcal{T})\ x = (A, \mathcal{T}) \wedge \mathcal{T} \subseteq \mathcal{P}(A) \wedge \emptyset, A \in \mathcal{T} \wedge (\forall U, V \in \mathcal{T})\ U \cap V \in \mathcal{T} \wedge (\forall \mathcal{W} \subseteq \mathcal{T})\ \bigcup \mathcal{W} \in \mathcal{T}$.

The last six examples illustrate something very important, namely, the great flexibility and succinctness of set theory as a *modeling language for all of mathematics*. Different *mathematical concepts* such as those of a graph, a poset, an abstract group, an abstract model of the natural numbers regardless of its representation, and a topological space are each easily axiomatized by an appropriate class as shown above. The general point is that the extensions of such mathematical concepts need not be sets, because the totality of the sets satisfying the requirements characterizing the given concept may be too big and therefore may be a *proper class*. Therefore, the concepts of graph, poset, group, representation-independent model of the natural numbers, and topological space are mathematically modeled not as sets, but as *classes*. Of course, some concepts, such as that of the empty set, may happen to be totalities that are actual sets describable by formulas in some appropriate definitional extension, in this example by the formula $x \in \emptyset$.

[3]The stated formula $\varphi(x)$ characterizes the class of triples that satisfy the *Peano-Lawvere* requirements for the natural numbers. The *Peano-Lawvere axiom* itself is then the sentence stating the *existence* of some such set, that is, the sentence $(\exists x)\ \varphi(x)$.

Exercise 109 *Write formulas in set theory axiomatizing the classes of: (i) abstract products; (ii) abstract coproducts; (iii) final sets; (iv) totally ordered sets; (v) lattices; (vi) chain-complete posets; and (vii) complete lattices.*

In a way analogous to sets, there is a containment relation between classes, so that some totalities are bigger than others. However, since classes are not sets, and in their *ZFC* formulation they are specified in an intensional way through formulas and not extensionally, the axiom of extensionality does not apply to them. Therefore, instead of an equality relation between classes, we must settle for an *equivalence* relation between classes, which in some sense captures indirectly the class extensionality idea. Specifically, given classes $\mathscr{A}$ and $\mathscr{B}$ we define the class containment relation $\mathscr{A} \subseteq \mathscr{B}$ by the defining equivalence

$$\mathscr{A} \subseteq \mathscr{B} \quad \textit{iff} \quad ZFC \vdash (\forall x)\ x : \mathscr{A} \ \Rightarrow\ x : \mathscr{B}.$$

We then define the equivalence relation between classes $\mathscr{A} \equiv \mathscr{B}$ by the defining equivalence

$$\mathscr{A} \equiv \mathscr{B} \quad \textit{iff} \quad \mathscr{A} \subseteq \mathscr{B} \ \textit{ and } \ \mathscr{B} \subseteq \mathscr{A}.$$

Some class containments are obvious. For example, any class $\mathscr{A}$ is contained in the universe $\mathscr{U}$. In particular, $\mathscr{U}^2 \subseteq \mathscr{U}$. It is also easy to see that for any $n \geq 2$ we have $\mathscr{U}^{n+1} \subseteq \mathscr{U}^n$, and that we have containments **Poset** $\subseteq$ **Graph** $\subseteq \mathscr{U}^2$, and **Fun** $\subseteq$ **Rel** $\subseteq \mathscr{U}$.

Exercise 110 *Prove the class equivalence*

$$(\forall x)\ \ x : \mathbf{Fun} \ \Leftrightarrow\ x : \mathbf{Rel} \wedge (\forall u, v, w)\ (((u, v) \in x \wedge (u, w) \in x) \Rightarrow v = w).$$

We can perform constructions on classes similar to set-theoretic constructions. For example, we can define union, intersection, complementation, difference, and cartesian product and cartesian power constructions by means of the defining equivalences:[4]

$$\begin{aligned}
x : (\mathscr{A} \cup \mathscr{B}) \quad &\Leftrightarrow \quad x : \mathscr{A} \ \vee\ x : \mathscr{B} \\
x : (\mathscr{A} \cap \mathscr{B}) \quad &\Leftrightarrow \quad x : \mathscr{A} \ \wedge\ x : \mathscr{B} \\
x : \overline{\mathscr{A}} \quad &\Leftrightarrow \quad \neg(x : \mathscr{A}) \\
x : (\mathscr{A} - \mathscr{B}) \quad &\Leftrightarrow \quad x : \mathscr{A} \ \wedge\ \neg(x : \mathscr{B}) \\
x : (\mathscr{A} \times \mathscr{B}) \quad &\Leftrightarrow \quad (\exists x_1, x_2)\ x_1 : \mathscr{A} \ \wedge\ x_2 : \mathscr{B} \ \wedge\ x = (x_1, x_2) \\
x : (\mathscr{A}^n) \quad &\Leftrightarrow \quad (\exists x_1, \ldots, x_n)\ x_1 : \mathscr{A} \ \wedge\ \ldots \wedge x_n : \mathscr{A} \ \wedge\ x = (x_1, \ldots, x_n).
\end{aligned}$$

where for any class $\mathscr{A}$ and any term t, the notation $t : \mathscr{A}$ abbreviates the formula $(\exists x)(t = x \wedge \mathscr{A})$. For example, $x_1 : \mathscr{A}$ abbreviates the formula $(\exists x)(x_1 = x \wedge \mathscr{A})$.

Since a class $\mathscr{A}$ is not a set, in general the subclasses of $\mathscr{A}$ are not sets either, so a powerclass construction $\mathcal{P}(\mathscr{A})$ is meaningless. There is, however, a perfectly meaningful construction $\mathcal{S}(\mathscr{A})$ of the class of all *subsets* of $\mathscr{A}$, namely,

$$x : \mathcal{S}(\mathscr{A}) \quad \Leftrightarrow \quad (\forall y \in x)\ y : \mathscr{A}.$$

By convention, given a set A and a class $\mathscr{A}$, we write $A \subseteq \mathscr{A}$ as an abbreviation for the formula $A : \mathcal{S}(\mathscr{A})$, and then say that A is *contained* in $\mathscr{A}$, or that A is a *subset* of $\mathscr{A}$.

16.1.1 Mathematical Theorems are Assertions about Classes

The above discussion about classes makes clear in which sense set theory is a universal modeling language for all of mathematics, and, through mathemathics, for science and technology. To drive the point home more forcefully, I want to briefly show how *theorems* in any given mathematical theory are also expressible as *conditional set theory formulas*, in which the condition makes explicit the theory for which the theorem holds; specifically, the theorem is made conditional to the *class* that axiomatizes the objects of the theory.

[4] In the equivalences below, $\mathscr{A}$ and $\mathscr{B}$ denote the predicate symbols used to define the corresponding classes. That is, we assume that we have defined, for example, the meaning of $\mathscr{A}$ by an equivalence $x : \mathscr{A} \Leftrightarrow \varphi(x)$.

Of course, in some cases the class of objects of a theory may happen to be set-defining, that is, such objects may range over a *set*. For example, the objects of the theory of numbers, or those of the elementary calculus, range, respectively, over the set of natural numbers $\mathbb{N}$, or over the set of real numbers $\mathbb{R}$ or the set of real-valued functions $[\mathbb{R}\to\mathbb{R}]$. Therefore, theorems about arithmetic or about the calculus are just set theory formulas in suitable definitional extensions of set theory. But such formulas are made *conditional* to the elements being natural numbers, or real numbers, or differentiable functions, and so on. For example, commutativity of natural number multiplication is a theorem of arithmetic stated by the set theory formula

$$(\forall n, m \in \mathbb{N})\ n * m = m * n$$

where the restricted quantification $(\forall n, m \in \mathbb{N})$ is a shorthand for making the equation conditional to n and m being natural numbers. Likewise, the *fundamental theorem of the calculus* can be formulated in a suitable definitional extension of set theory as the formula

$$(\forall a, b \in \mathbb{R})(\forall f \in [\mathbb{R}\to\mathbb{R}])\ (a < b \ \wedge\ \mathit{Diff}(f)) \ \Rightarrow \int_a^b f'(x) \cdot dx = f(b) - f(a)$$

where $\mathit{Diff}(f)$ is a unary predicate axiomatizing the differentiable functions, f' is the derivative of f, and where, again, all is made conditional to $a, b \in \mathbb{R}$ and $f \in [\mathbb{R}\to\mathbb{R}]$.

However, the objects of many mathematical theories, for example, the theory of partial orders, the theory of abstract groups, and of course the theory of sets itself, do not range over a set but over a *proper class*, such as, for example, the class **Poset** of all posets, the class **Group** of all groups, or the universe class $\mathscr{U}$ of all sets. In this case, the theorems of the given theory become *conditional statements* in the language of set theory, where the theorem is asserted not for all sets, but for those sets in the given class. For example, in the theory of posets, the *duality theorem*, stating that for any poset the reverse relation is also a poset is a conditional statement of the form

$$(\forall A, R)\ (A, R) : \mathbf{Poset} \ \Rightarrow\ (A, R^{-1}) : \mathbf{Poset}.$$

Similarly, in the theory of groups, the theorem stating that all groups satisfy the equation $e \cdot y = y$ is a conditional statement of the form

$$(\forall G, e, i, m)\ (G, e, i, m) : \mathbf{Group} \ \Rightarrow\ (\forall y \in G)\ m(e, y) = y.$$

Of course, for the theory of sets itself the statements become *unconditional*, since all sets belong to the universe $\mathscr{U}$. For example, Exercise 1, together with the extensionality axiom amount to a theorem stating that the pair $(\mathscr{U}, \subseteq)$ is a poset or, more properly speaking but also more awkwardly, a "poclass." That is, although $\mathscr{U}$ is not a set, the intensional relation $\subseteq$ does however satisfy all the axioms of partially ordered "sets." And this result plus Exercises 7 and 8 amount to a proof of another theorem stating that $(\mathscr{U}, \subseteq)$ is "almost a lattice," in the sense that it satisfies all the lattice axioms except for the existence of a top element. Again, now the axioms are asserted not about a set with an ordering, but about the universe class $\mathscr{U}$ with the intensional containment relation $\subseteq$ as its ordering.

16.2 Intensional Relations

Both relations and functions can be understood in two, quite different ways: intensionally, as *formal specifications* in an appropriate *language* (a programming language if we want the relation or function to be computable, or, here, the formal language of set theory), and extensionally, as characterized by appropriate *sets*. The most intuitive and historically oldest way is in fact the the *intensional* way. In the intensional sense, a relation is a *precise linguistic specification* of how two things are related. For example, we can say that x and y are related, with x and y real numbers, iff $x = y^2$. This is a precise, linguistic specification of the relation: "y is the square root of x." This is a relation and not a function, since in general there may be more than one y (or maybe none) in that relation to a given x. For example, if $x = 4$, then y can be either 2 or -2; and if $x = -1$, there is no such y if we assume that y must be a real number. In general, *any* set theory formula φ whose set of free variables is a subset of $\{x, y\}$ can be used to specify a relation intensionally. Then, given sets A and B, the *extensional* interpretation of such a relation is the set

$$\{(x, y) \in A \times B \mid \varphi\}.$$

For example, for the sets $A = \mathbb{R}$ and $B = \mathbb{R}$, the extensional interpretation of the intensional relation $x = y^2$ is the square root relation

$$SQRT = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x = y^2\}.$$

In this sense, the notion of relation studied in §6 is an *extensional* notion, corresponding to a *set* of pairs. But an intensional relation gives us a much more general notion of relation than any particular set extension we could associate to it. That is, the relational notion can be intrinsically *more general* than any of its possible set extensions. For example, we can consider the intensional relation $x = y^2$, with x and y ranging over all *cardinals*, which by Lemma 15 are a proper class (see §18.5.1 for a precise definition of this class). Or consider the example of the set containment relation $x \subseteq y$. This relation makes sense for *any pair of sets* x and y. Note that, if we wish, the binary predicate symbol $\subseteq$ could be *defined away*, since it is defined by the equivalence

$$x \subseteq y \;\Leftrightarrow\; (\forall z)\, z \in x \;\Rightarrow\; z \in y.$$

In other words, the containment relation makes sense not just for any pair of subsets of a given set A, where we would give it the extension

$$\{(x, y) \in \mathcal{P}(A) \times \mathcal{P}(A) \mid x \subseteq y\}$$

but for any pair of sets x and y whatsoever in the *class* of sets $\mathscr{U}$. In this sense, intensional relations, as formulas $\varphi(x, y)$ with variables among x, y, are also analogous to Balinese shadow theater puppets, since they are a small piece of language specifying a potentially huge subclass of $\mathscr{U}^2$ (namely, the subclass defined by the formula $(\exists y, z)\; x = (y, z) \;\wedge\; \varphi(y, z)$), which in general need not be a set.

Definition 26 *An* intensional relation *is a formula φ in the formal language of set theory, or in a definitional extension*[5] *of the basic set theory language, such that its set of free variables is contained in $\{x, y\}$. Such an intensional relation φ can have many* set extensions*. Specifically, for any two sets A and B, its set extension relative to A and B is the set of pairs*

$$\{(x, y) \in A \times B \mid \varphi\}.$$

It has also an unrestricted interpretation *as the class* $(\exists y, z)\; x = (y, z) \;\wedge\; \varphi(y, z)$.

As for classes, we can define a notion of containment between intensional relations φ and ψ by the defining equivalence

$$\varphi \subseteq \psi \quad \textit{iff} \quad ZFC \vdash (\forall x, y)\; \varphi \Rightarrow \psi.$$

We can then define an equivalence relation $\varphi \equiv \psi$ between intensional relations φ and ψ by means of the defining equivalence

$$\varphi \equiv \psi \quad \textit{iff} \quad \varphi \subseteq \psi \;\; \textit{and} \;\; \psi \subseteq \varphi.$$

For example, we can define the intensional identity relation $x\, id\, y$ by the formula $x = y$, and the intensional strict containment relation $x \subset y$ by the formula $x \subseteq y \;\wedge\; x \neq y$. It is then trivial to check that we have the containments of intensional relations $x\, id\, y \;\subseteq\; (x \subseteq y)$, and $(x \subset y) \;\subseteq\; (x \subseteq y)$. These two containment are interesting, because both make clear that the symbol "$\subseteq$" is used here in *two very different ways*: (i) as a binary predicate symbol in our extended set theory language in the formula $x \subseteq y$; and (ii) as an abbreviated *meta-notation* for relating two intensional relations φ and ψ by the containment relation $\varphi \subseteq \psi$ iff we can *prove* the formula $(\forall x, y)\; \varphi \Rightarrow \psi$.

Language is quite wonderful, in that we can use it to *compute* things with it. For example, given an intensional relation $\varphi(x, y)$, we can *define* its *inverse* relation $\varphi(x, y)^{-1}$ by the defining identity:

$$\varphi(x, y)^{-1} = \varphi(y, x).$$

For example, the inverse relation of the square root relation $x = y^2$ is the relation $y = x^2$; and the inverse relation of the containment relation $x \subseteq y$ is the relation $y \subseteq x$, which is typically abbreviated by $x \supseteq y$. We can also *compose* intensional relations. That is, given intensional relations φ and ψ, we can define their *composition* $\varphi\,;\psi$ by means of the defining equivalence

$$\varphi\,;\psi \;\; \textit{iff} \;\; (\exists z)\; \varphi(x, z) \;\wedge\; \psi(z, y).$$

[5] We will of course allow ourselves the freedom of extending our set theory language with new function and predicate symbols, such as $\subseteq$ above. However, all such symbols can be *defined away* by applying their corresponding definitions (as illustrated for $\subseteq$ above), so that in the end, if we wish, we can obtain an equivalent formula in the basic language of set theory presented in §2.

For example, it is easy to prove that the composed intensional relation $(x \subset y); (y \subset x)$ is equivalent to each of the (always false) intensional relations $x \neq x$ and $x \in \emptyset$; that is, we can prove the equivalences of intensional relations $(x \subset y); (y \subset x) \equiv x \neq x \equiv x \in \emptyset$.

Exercise 111 *(Analogous to Lemma 4 for intensional relations). Prove that for any intensional relations φ, ψ and ϕ, we have equivalences:*

- $(\varphi\,;\psi);\phi \;\equiv\; \varphi\,(\psi\,;\phi)$
- $\varphi\,;id \;\equiv\; \varphi$
- $id\,;\varphi \;\equiv\; \varphi$.

16.3 Intensional Functions

What is an *intensional function*? It is just an intensional relation φ such that we can *prove* in set theory that φ assigns a unique y to each x. This can be easily generalized to the notion of *intensional function of several arguments* as follows.

Definition 27 *An* intensional function *(of one argument) is an intensional relation φ such that we can prove*

$$ZFC \vdash (\forall x)(\exists! y)\ \varphi.$$

For $n \geq 2$, an intensional function of n arguments *is a formula φ in set theory whose free variables are contained in the set $\{x_1, \ldots, x_n, y\}$, and such that we can prove*

$$ZFC \vdash (\forall x_1, \ldots, x_n)(\exists! y)\ \varphi.$$

For example, the union of two sets is a binary intensional function defined by the formula, $(\forall z)\ z \in y \;\Leftrightarrow\; z \in x_1 \vee z \in x_2$.

Note that we can *always* express an intensional function of one or more arguments in so-called *term form* as a formula of the form $y = t(x)$, or $y = t(x_1, \ldots, x_n)$, provided we allow ourselves the freedom of using a suitable *definitional extension* of our set theory language. As already indicated in §6.2 (where term forms were favoured for function definitions), the great advantage of having an intensional function in term form $y = t(x)$ is that then the proof of the formula $(\forall x)(\exists! y)\ \varphi$ becomes *trivial*, and therefore can be omitted.

The reason why we can always express an intensional function in term form is as follows. Given our intensional function $\varphi(x, y)$ (resp., $\varphi(x_1, \ldots, x_n, y)$) two things can happen: (i) either we can find a term $t(x)$ (resp., $t(x_1, \ldots, x_n)$) such that in our current definitional extension of set theory we can prove the equivalence $\varphi \equiv (y = t(x))$ (resp., $\varphi \equiv (y = t(x_1, \ldots, x_n))$) and then we can already express our function in term form; or (ii) we cannot find such a term, in which case we can introduce in our set theory language a *new* function symbol, say f, of one argument (resp., n arguments) and extend our set theory language with the new definitional extension

$$y = f(x) \;\Leftrightarrow\; \varphi(x, y)$$

resp.,

$$y = f(x_1, \ldots, x_n) \;\Leftrightarrow\; \varphi(x_1, \ldots, x_n, y).$$

Such definitional extension is *correct* (i.e., does not mess up our set theory in any way) precisely because we have already *proved* the formula $(\forall x)(\exists! y)\ \varphi$ (resp., $(\forall x_1, \ldots, x_n)(\exists! y)\ \varphi$). For example, if we had not yet introduced the binary union symbol $\_ \cup \_$ in our language, we could do so by means of the definitional extension

$$y = x_1 \cup x_2 \;\Leftrightarrow\; (\forall z)\ z \in y \;\Leftrightarrow\; z \in x_1 \vee z \in x_2.$$

Once we have defined the union $\_ \cup \_$ and pairing $\{\_, \_\}$ functions in this definitional extension way, we could also define the intensional successor function in term form by the formula $y = x \cup \{x\}$. Of course, if we insist,we can always *define away* all function symbols, and express any intensional function as a formula in the basic language of set theory presented in §2.

We can now better understand that our formula, assignment, and lambda notations for extensional relations and functions introduced in §6.2 were just *intensional* relations and functions used to *specify* their

corresponding extensional relations and functions, once we had chosen appropriate sets A and B for their domain and codomain. But these notations also make complete sense at the much more general intensional level. For intensional relations this is obvious, since we have defined them as formulas. For intensional functions, they can also be specified as formulas for which we can prove the uniqueness of their y result. But if we have expressed a function in term form $y = t(x)$ (resp., $y = t(x_1, \ldots, x_n)$), then we can, equivalently, describe it in *assigment notation* as $x \mapsto t(x)$ (resp., $(x_1, \ldots, x_n) \mapsto t(x_1, \ldots, x_n)$); and in *lambda notation* as $\lambda x.\ t(x)$ (resp., $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$).

Furthermore, specifying intensional functions in lambda notation frees us not only from having to mention the output variable y, but also from the requirement of having to use *exactly* x (resp., $(x_1, \ldots, x_n)$) as the input variable (resp., tuple of input variables). We can instead use *any* name (resp., any tuple of names) for the input variable (resp., tuple of input variables). This is because the lambda notation makes it unambiguous which is the input variable, and when there is a tuple of variables, which is the *order* of the arguments indicated by the variables. For example, the intensional function $\lambda(x_1, x_2).\ [x_1 \to x_2]$ can instead be unambiguously defined by the equivalent lambda expression $\lambda(A, B).\ [A \to B]$. This equivalence between lambda expressions up to renaming of variables is called *α-equivalence*.

Exercise 112 *(Equivalence of intensional functions). Let $f = \lambda x.\ t(x)$ and $g = \lambda x.\ t'(x)$ be two intensional functions. Then, by definition, $f \equiv g$ iff f and g are equivalent as intensional relations, that is, iff ZFC $\vdash (y = t(x)) \Leftrightarrow (y = t'(x))$. Prove using elementary logical reasoning that $f \equiv g$ iff ZFC $\vdash (\forall x)\ t(x) = t'(x)$. This shows that the notion of equivalence of intensional functions is the natural one: if for all inputs two intensional functions provide the same outputs, then they should be considered equivalent. Generalize this to intensional functions of several variables. Prove the equivalence of intensional functions $(\lambda(x_1, x_2).\ x_1 \cup x_2) \equiv (\lambda(x_1, x_2).\ x_2 \cup x_1)$.*

16.3.1 Typing Intensional Functions

In §6.2 we already encountered the problem that a function specification $\lambda x \in A.\ t(x) \in B$ might not really define a function from A to B but only a *partial* function from A to B, unless we could *prove* the formula $(\forall x \in A)\ t(x) \in B$. That is, unless we could correctly *type* the intensional function $\lambda x.\ t(x)$ as a function of the form $\lambda x \in A.\ t(x) \in B$. We can now broaden our notion of typing for intensional functions in a very general way:

Definition 28 *Given an intensional function $\lambda x.\ t(x)$, and given classes $\mathscr{A}$, $\mathscr{B}$, and sets A and B, we say that $\lambda x.\ t(x)$ has:*

- class-to-class typing $\lambda x{:}\mathscr{A}.\ t(x){:}\mathscr{B}$ *iff* ZFC $\vdash (\forall x)(x{:}\mathscr{A} \Rightarrow t(x){:}\mathscr{B})$.
- set-to-class typing $\lambda x \in A.\ t(x){:}\mathscr{B}$ *iff* ZFC $\vdash (\forall x)(x \in A \Rightarrow t(x){:}\mathscr{B})$.
- class-to-set typing $\lambda x{:}\mathscr{A}.\ t(x) \in B$ *iff* ZFC $\vdash (\forall x)(x{:}\mathscr{A} \Rightarrow t(x) \in B)$.
- set-to-set typing $\lambda x \in A.\ t(x) \in B$ *iff* ZFC $\vdash (\forall x \in A)\ t(x) \in B$.

Similarly, given an intensional function $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$, and given classes $\mathscr{A} \subseteq \mathscr{U}^n$, $\mathscr{B}$, and sets $A_1, \ldots, A_n$ and B, we say that $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$ has:

- class-to-class typing $\lambda(x_1, \ldots, x_n){:}\mathscr{A}.\ t(x){:}\mathscr{B}$ *iff* ZFC $\vdash (\forall x, x_1, \ldots, x_n)((x{:}\mathscr{A} \wedge x = (x_1, \ldots, x_n)) \Rightarrow t(x_1, \ldots, x_n){:}\mathscr{B})$.
- set-to-class typing $\lambda(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n.\ t(x) : \mathscr{B}$ *iff* ZFC $\vdash (\forall x_1, \ldots, x_n)((x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \Rightarrow t(x_1, \ldots, x_n){:}\mathscr{B})$.
- class-to-set typing $\lambda(x_1, \ldots, x_n){:}\mathscr{A}.\ t(x) \in B$ *iff* ZFC $\vdash (\forall x, x_1, \ldots, x_n)((x{:}\mathscr{A} \wedge x = (x_1, \ldots, x_n)) \Rightarrow t(x_1, \ldots, x_n) \in B)$.
- set-to-set typing $\lambda(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n.\ t(x) \in B$ *iff* ZFC $\vdash (\forall x_1, \ldots, x_n)((x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \Rightarrow t(x_1, \ldots, x_n) \in B)$.

If we denote by f such an intensional function, then for each of the respective typings we then can also use the notation: $f : \mathscr{A} \longrightarrow \mathscr{B}$, $f : A \longrightarrow \mathscr{B}$, $f : \mathscr{A} \longrightarrow B$, and $f : A \longrightarrow B$. And for f of n arguments the notation: $f : \mathscr{A} \longrightarrow \mathscr{B}$, $f : A_1 \times \ldots \times A_n \longrightarrow \mathscr{B}$, $f : \mathscr{A} \longrightarrow B$, and $f : A_1 \times \ldots \times A_n \longrightarrow B$.

For example, the successor function $s = \lambda x.\ x \cup \{x\}$ has the class-to-class typing $s : \mathscr{U} \longrightarrow x \neq \emptyset$, and the set-to-set typing $s : \mathbb{N} \longrightarrow \mathbb{N}$. Similarly the functions $p_1 = \lambda(x_1, x_2).\ x_1$, and $p_2 = \lambda(x_1, x_2).\ x_2$ have the class-to-class typings $p_1, p_2 : \mathscr{U}^2 \longrightarrow \mathscr{U}$, and for any sets A_1 and A_2 the set-to-set typings $p_1 : A_1 \times A_2 \longrightarrow A_1$, and $p_2 : A_1 \times A_2 \longrightarrow A_2$. Finally, the cardinality function $|\_| = \lambda x.\ |x|$ has the class-to-class typing $|\_| : \mathscr{U} \longrightarrow$ **Card** (where **Card** is the class of cardinals that will be defined in detail in §18.5.1), the class-to-set typing $|\_| : \mathscr{U}_{fin} \longrightarrow \mathbb{N}$, and the set-to-set typing $|\_| : \mathbb{N} \longrightarrow \mathbb{N}$.

Note that, given intensional functions $f = \lambda x.\ t(x)$, and $g = \lambda x.\ t'(x)$ with respective typings $f : \mathscr{A} \longrightarrow \mathscr{B}$, and $g : \mathscr{A} \longrightarrow \mathscr{C}$, we can define the intensional function $(f, g) = \lambda x.\ (t(x), t'(x))$ with typing $(f, g) : \mathscr{A} \longrightarrow \mathscr{B} \times \mathscr{C}$. We can also mix and match set and class typings in a similar way. For example, if f has typing $f : A \longrightarrow \mathscr{B}$, and g has typing $g : A \longrightarrow C$, then (f, g) has typing $(f, g) : A \longrightarrow \mathscr{B} \times C$, where, the class $\mathscr{B} \times C$ is defined by the equivalence: $x : (\mathscr{B} \times C) \Leftrightarrow (\exists y, z)\ x = (y, z) \wedge y : \mathscr{B} \wedge z \in C$. The function (f, g) is defined in a similar way when f and g are intensional functions of n arguments. Furthermore, the construction (f, g) generalizes easily to the construction $(g_1, \ldots, g_n)$ for n intensional functions $g_1, \ldots, g_n$ with $n \geq 2$.

Let us call an intensional function f defined by a formula $\varphi(x, y)$ *injective*, resp., *surjective*, *for the typing* $f : \mathscr{A} \longrightarrow \mathscr{B}$, iff f has that typing and, furthermore, $ZFC \vdash (\forall x, x', y)\ x : \mathscr{A} \wedge x' : \mathscr{A} \wedge \varphi(x, y) \wedge \varphi(x', y) \Rightarrow x = x'$, resp., $ZFC \vdash (\forall y)\ y : \mathscr{B} \Rightarrow (\exists x)\ x : \mathscr{A} \wedge \varphi(x, y)$. Similarly, we call such an f *bijective* for the typing $f : \mathscr{A} \longrightarrow \mathscr{B}$ iff f is both injective and surjective for that typing. As done for extensional functions, we will write $f : \mathscr{A} \rightarrowtail \mathscr{B}$, resp., $f : \mathscr{A} \twoheadrightarrow \mathscr{B}$, to indicate that f is injective, resp., surjective, for that typing. As a special case, we call an intensional function f *injective*, resp., *surjective*, resp., *bijective*, iff it is so for the typing $f : \mathscr{U} \longrightarrow \mathscr{U}$. For example, the intensional function $\lambda x.\ \{x\}$ is injective, and is bijective for the typing $\lambda x.\ \{x\} : \mathscr{U} \longrightarrow$ **Sing**.

Exercise 113 *(Composing Intensional Functions). Since any intensional functions f and g of one argument are intensional relations, their composition $f; g$ is always defined as an intensional relation. Prove that if f, g are intensional functions, then $f; g$ is always an intensional function. Similarly, if f is an intensional function of n arguments ($n \geq 1$) defined by the formula $\varphi(x_1, \ldots, x_n, y)$, and $g_1, \ldots, g_n$ are intensional functions of m arguments ($m \geq 1$) defined by the formulae $\psi_j(x_1, \ldots, x_m, y)$, $1 \leq j \leq m$, then the composition $(g_1, \ldots, g_n); f$ can be defined by the formula $(\exists z_1, \ldots, z_n)\ \psi_1(x_1, \ldots, x_m, z_1) \wedge \ldots \psi_n(x_1, \ldots, x_m, z_n) \wedge \varphi(z_1, \ldots, z_n, y)$ and is also a function. Prove also that the function $(g_1, \ldots, g_n)$ defined above as a lambda expression, assuming that the $g_1, \ldots, g_n$ were expressed in term form, is equivalent to the definition of $(g_1, \ldots, g_n)$ as the formula $(\exists z_1, \ldots, z_n)\ \psi_1(x_1, \ldots, x_m, z_1) \wedge \ldots \psi_n(x_1, \ldots, x_m, z_n) \wedge (z_1, \ldots, z_n) = y$. Finally, prove that if f and g have typings $f : \mathscr{X} \longrightarrow \mathscr{Y}$, and $g : \mathscr{Y} \longrightarrow \mathscr{Z}$, where the $\mathscr{X}$, $\mathscr{Y}$, and $\mathscr{Z}$ could be classes or sets, then $f; g$ has typing $f; g : \mathscr{X} \longrightarrow \mathscr{Z}$.*

Exercise 114 *(***if**$\_ : \mathscr{A}$ **then_else_fi***, generalizes Exercise 42). Given any class $\mathscr{A}$, define formally an intensional function of three arguments, let us call it* **if**$\_ : \mathscr{A}$ **then_else_fi***, such that given arguments (x_1, x_2, x_3) if $x_1 : \mathscr{A}$, then the result is x_2, and otherwise the result is x_3.*

Exercise 115 *(Extending partial intensional functions to total ones). Call an intensional relation $\varphi(x, y)$ a* partial intensional function *iff $ZFC \vdash (\forall x, y, z)\ \varphi(x, y) \wedge \varphi(x, z) \Rightarrow y = z$. Give an explicit construction associating to any such partial intensional function $\varphi(x, y)$ a new formula $\varphi'(x, y)$ such that: (i) $\varphi'(x, y)$ is a (total) intensional function; and (ii) $\varphi'(x, y)$ extends $\varphi(x, y)$, in the precise sense that we can prove $ZFC \vdash \varphi(x, y) \Rightarrow \varphi'(x, y)$. Use your construction to extend an extensional function $f : A \longrightarrow B$, where A, B, and f have been specified by suitable terms in a definitional extension of set theory, to an intensional function $f : \mathscr{U} \longrightarrow \mathscr{U}$.*

16.3.2 Computing with Intensional Functions

Lambda notation is very convenient, since it was developed explicitly as a notation for computing with intensional functions. In particular, lambda notation makes it easy to *symbolically evaluate* an intensional function on an input term. This is accomplished by using an *application notation*[6] $u(v)$, which is a new term obtained from an intensional function definition u and a term v. Intuitively, $u(v)$ symbolically describes the application of the intensional function u to the argument v. For example, if $u = \lambda x.\ [2 \rightarrow x]$, and $v = \mathbb{N}$, $u(v)$ is the application term $u(v) = (\lambda x.\ [2 \rightarrow x])(\mathbb{N})$.

[6] In the lambda calculus, both arguments of an application can be lambda terms. That is, we can apply an intensional function to another intensional function. However, for our present purposes it is enough to consider this more restricted form of application, in which a *lambda expression* is applied to a *term*, whose function symbols belong to some definitional extension of set theory.

Symbolic evaluation, to compute the resulting *term* denoted by $u(v)$, is then accomplished by the so-called β *rule* of the lambda calculus, which is the equation:

$$(\beta) \qquad (\lambda X.\ t)(v) = t\{X \mapsto v\}$$

where t stands for any term having at most the single variable X, and $t\{X \mapsto v\}$ (sometimes abbreviated to just $t(v)$) denotes the result of *replacing* each occurrence[7] of X in t by v, where X is a *schematic variable*, which stands not for any concrete variable, but for whichever variable may have been bound by the lambda in the concrete application expression. What the β-rule tells us is that to symbolically evaluate an application expression $(\lambda x.\ t)(v)$, where the intensional function $\lambda x.\ t$ is applied to the argument v, what we do is to *syntactically replace* all occurrences of x in t by v. For example, our application term $u(v) = (\lambda x.\ [2{\to}x])(\mathbb{N})$ evaluates in this way to $[2{\to}\mathbb{N}]$ by replacing x by $\mathbb{N}$ in the term $[2{\to}x]$.

In a similar way, we can introduce an application notation to evaluate intensional functions of several arguments. Given an intensional function $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$ and terms $v_1, \ldots, v_n$, we obtain the application expression $(\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n))(v_1, \ldots, v_n)$ which is evaluated according to the n-argument β-rule

$$(\beta)_n \qquad (\lambda(X_1, \ldots, X_n).\ t)(v_1, \ldots, v_n) = t(v_1, \ldots, v_n)$$

where $t(v_1, \ldots, v_n)$ is an abbreviation for the more precise simultaneous substitution[8] $t(X_1, \ldots, X_n)\{X_1 \mapsto v_1, \ldots, X_n \mapsto v_n\}$. For example, given the lambda term $\lambda(x_1, x_2).\ [x_2{\to}\mathcal{P}(x_1)]$, and given the terms $\mathbb{N}$ and 7, we can form the application expression $\lambda(x_1, x_2).\ [x_2{\to}\mathcal{P}(x_1)](\mathbb{N}, 7)$, which is symbolically evaluated as follows:

$$\lambda(x_1, x_2).\ [x_2{\to}\mathcal{P}(x_1)](\mathbb{N}, 7) = [7{\to}\mathcal{P}(\mathbb{N})].$$

Exercise 116 *Use the (β) and $(\beta)_n$ rules to symbolically evaluate the application expressions $\lambda x.\ x(3)$, $\lambda(x_1, x_2).\ x_1(2, 3)$, and $\lambda x.x \cup \{x\}(4)$.*

Exercise 117 *($(\beta)_n$ as a special case of (β)). Show that an n-ary intensional function $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$ can always be expressed as a function of one argument, namely, as the function $\lambda x.\ t(p_1(x), \ldots, p_n((x))$, where p_i is the i-th projection function $\lambda(x_1, \ldots, x_n).\ x_i$, which we may assume has been defined by a* unary *function symbol p_i in a suitable definitional extension of set theory (since $\mathscr{U}^n \subseteq \mathscr{U}$, when the argument x of p_i is not an n-tuple, then $p_i(x)$ can be defined to evaluate to a fixed chosen value such as, e.g., $\emptyset$). In particular, since given terms $v_1, \ldots, v_n$, the tuple $(v_1, \ldots, v_n)$ is also a term, we can apply to it both the function $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$, and the function $\lambda x.\ t(p_1(x), \ldots, p_n(x))$. Show that then the $(\beta)_n$ rule (resp., (β) rule) evaluate these two function applications to provably equal terms, so $(\beta)_n$ can be viewed as a special case of (β).*

Exercise 118 *(Function Application and Typing). Suppose that the intensional function $f = \lambda x.\ t$ has the typing $f : \mathscr{X} \longrightarrow \mathscr{Y}$, where the $\mathscr{X}$ and $\mathscr{Y}$ could be classes or sets. Prove that if the term v is such that $v : \mathscr{X}$ (resp., $v \in \mathscr{X}$), then $f(v) = \lambda x.\ t(v) = t\{x \mapsto v\}$ is such that $t\{x \mapsto v\} : \mathscr{Y}$ (resp., $t\{x \mapsto v\} \in \mathscr{Y}$). State and prove the analogous result for an n-argument function $f = \lambda(x_1, \ldots, x_n).\ t$.*

Exercise 119 *(Function Composition as Evaluation). Suppose that we have intensional functions $f = \lambda x.\ t(x)$ and $g = \lambda y.\ t'(y)$. Prove that their function composition $f; g$ is equivalent to the intensional function defined by $\lambda x.\ t'\{y \mapsto t(x)\}$. State and prove the analogous result for the composition $(g_1, \ldots, g_n); f$, where f is an intensional function of n arguments ($n \geq 1$), and the $g_1, \ldots, g_n$ are intensional functions of m arguments ($m \geq 1$), showing the equivalence of the intensional function defined by your resulting lambda expression with the definition of $(g_1, \ldots, g_n); f$ by the formula in Exercise 113.*

16.3.3 Dependent and Polymorphic Types

Set-to-class intensional functions provide a semantics for *dependent data types* in programming languages, where the set which is the domain of the intensional function is the parameter set over which the dependent type "depends." Similarly, the notion of a class-to-class intensional function is very useful to give a precise semantics to so-called *polymorphic types* (also called *generic types*) in a programming language. Both

[7] A variable x may occur several times in a term t, and then all those occurrences of x must be *simultaneously* replaced by v. For example, if $t = x + x$ and $v = y \cdot x$, then $t\{x \mapsto (y \cdot x)\}$ is the term $(y \cdot x) + (y \cdot x)$.

[8] That is, we simultaneously replace each occurence of X_1 in t by v_1, each occurence of X_2 in t by v_2, ..., and each occurrence of X_n in t by v_n. For example, if $t = x + (y * x)$, then $t\{x \mapsto x + z, y \mapsto y * z\} = (x + z) + ((y * z) * (x + z))$ (see Definition 5).

dependent and polymorphic types are *parameterized* types. The difference is that in a dependent type the parameter ranges over a *set*, whereas in a polymorphic type it ranges over a *class*. However, as we shall see, we may also have "mixed" parameterized types, which may be *dependent* in some parameters, and *polymorphic* on other parameters.

A well-known example of dependent type, already discussed in §12, is the dependent type of arrays of rational numbers. We can naturally express this dependent type as a set-to-class intensional function of the form $Array(\mathbb{Q}) = \lambda n \in \mathbb{N}.\ [n{\to}\mathbb{Q}] : \mathscr{U}$. By Exercise 82 we know that $Array(\mathbb{Q})$ can also be typed as a set-to-set surjective function, giving us an $\mathbb{N}$-indexed set $Array(\mathbb{Q}) = \{[n{\to}\mathbb{Q}]\}_{n\in\mathbb{N}}$ in the sense of §12. In general, however, the notion of a set-to-class intensional function provides a *different* and alternative notion of "family of sets," than that characterized in §12 as a surjective extensional function. The adequate way to relate both notions is by means of the Axiom of Replacement, as explained in §16.4.

But why stopping at arrays of rational numbers? Why not considering arrays for *any* set X of data elements? In this way we obtain a data type

$$Array = \lambda(X, n) : \mathscr{U} \times \mathbb{N}.\ [n{\to}X] : \mathscr{U}$$

which is *polymorphic* on its first parameter X, and *dependent* on its second parameter n. Note that when we apply the intensional function $Array$ to any pair (B, n) with B a set and n a natural number, then the β rule gives us the resulting type $Array(B, n) = [n{\to}B]$.

Note, also, that if we instantiate the parameter X of the, "mixed" polymorphic and dependent, type $Array$ to a given set B, we obtain the purely dependent type

$$Array(B) = \lambda n \in \mathbb{N}.\ [n{\to}B] : \mathscr{U}$$

which for different choices of B gives us our original example $Array(\mathbb{Q})$, and also, say, $Array(\mathbb{N})$, $Array(\mathbb{Z})$, $Array(2)$, and so on.

In general, a *polymorphic data type* T is just a class-to-class function $T : \mathscr{A} \longrightarrow \mathscr{B}$ for suitable classes $\mathscr{A}$ and $\mathscr{B}$, where we may have $\mathscr{A} \subseteq \mathscr{U}^n$, and therefore the polymorphic type can have several parameters. For example, the *polymorphic* data type $List(X)$ of lists formed from elements in the parameter set X is the class-to-class intensional function

$$List = \lambda X : \mathscr{U}.\ \bigcup Array(X) : \mathscr{U}.$$

Similarly, the *polymorphic* product $X_1 \times X_2$ and disjoint union $X_1 \oplus X_2$ data types are the class-to-class intensional functions

$$\times = \lambda(X_1, X_2) : \mathscr{U}^2.\ X_1 \times X_2 : \mathscr{U}.$$

and

$$\oplus = \lambda(X_1, X_2) : \mathscr{U}^2.\ X_1 \oplus X_2 : \mathscr{U}.$$

In some cases, the class over which the parameters of the polymorphic type range can be more sophisticated than just $\mathscr{U}$ or $\mathscr{U}^n$. For example, a polymorphic *sorting module* $Sort(P)$ has a parameter P ranging over the class **Toset** of totally ordered sets, that is, it is a class-to-class intensional function of the form

$$Sort = \lambda P : \mathbf{Toset}.\ Sort(P) : \mathscr{U} \times \mathbf{Fun}$$

where $Sort(A, \leq)$ is a pair of the form $Sort(A, \leq) = (List(A), sort(A, \leq))$, where $List(A)$ is the already-described type of lists with elements in A, and $sort(A, \leq)) : List(A) \longrightarrow List(A)$ is the sorting function associated to the totally ordered set $(A, \leq)$.

16.4 The Axiom of Replacement

As hinted at in §12, and mentioned also above in our discussion of polymorphic and dependent types, given an index set I we have two *diferent* notions or concepts of "I-indexed family of sets," that are both quite intutive and natural. On the one hand, we have the *extensional* notion of such a family as a surjective function $A : I \twoheadrightarrow A[I]$ defined in detail in §12. On the other, we also have the *intensional* notion given by a set-to-class intensional function $A : I \longrightarrow \mathscr{B}$. The key issue, as already pointed out in §12, is that if we

are given an intensional function $A : I \longrightarrow \mathscr{B}$, we may have in principle no clue about how the sets $A(i)$ for each $i \in I$ are all elements of some common *set* T. Put otherwise, we may not have any idea about how to *type* the set-to-class intensional function $A : I \longrightarrow \mathscr{B}$ as a set-to-set intensional function $A : I \longrightarrow T$.

This is a nontrivial matter, and a genuine concern. Given a set-to-class intensional function $A : I \longrightarrow \mathscr{B}$, does a set T always exist so that we can type A as a set-to-set intensional function $A : I \longrightarrow T$? If it does, then our two, intensional and extensional, notions of "I-indexed family of sets" become reconciled, since the intensional notion then becomes a *linguistic specification* for the extensional one. That is, we can then define the extensional I-indexed set specified by the intensional function $A : I \longrightarrow \mathscr{B}$ as the surjective function $A : I \twoheadrightarrow A[I] : i \mapsto A(i)$, where $A[I] = \{A(i) \in T \mid i \in I\}$. This seems plausible. For example, for the set-to-class intensional successor function $\lambda x \in \mathbb{N}.\ x \cup \{x\} : \mathscr{U}$ such a set T exists, namely, $T = \mathbb{N} - \{0\}$, so that we have a corresponding surjective function $s : \mathbb{N} \twoheadrightarrow (\mathbb{N} - \{0\})$. Similarly, for the array of rationals set-to-class function $Array(\mathbb{Q}) = \lambda n \in \mathbb{N}.\ [n{\to}\mathbb{Q}] : \mathscr{U}$, Exercise 82 ensures that such a set T exists. But what about other examples?

This is, again, another walk on the tight rope of restricted comprehension. Intuitively, it is all a matter of how *big* such a set $\{A(x) \mid x \in I\}$ can be. If it is too big, we would run against Russell-like paradoxes and all would be lost. But since I is a set, and therefore of bounded size, intuitively the size of the set $\{A(x) \mid x \in I\}$ cannot be bigger[9] than that of I. So, assuming that those sets exist seems safe. Yet, unless we give an explicit axiom in our set theory, we cannot ensure their existence. This is exactly what the *axiom scheme of replacement* does. In plain English it says:

Any set-to-class intensional function has a set-to-set typing.

A first attempt to capture this formally as an axiom scheme is to make it range over all functional terms t having a single variable x in a chosen definitional extension of set theory:

$$(Rep_0) \quad (\forall I)(\exists! T)(\forall u)\ (u \in T \ \Leftrightarrow\ (\exists i \in I)\ u = t(i))$$

where the unique set T, asserted to exist given I, could be denoted as $\{t(i) \mid i \in I\}$, or in more abbreviated form as $t[I]$. However, this formulation, although quite intutive and in practice the one we will often use, is still *too weak*, and also somewhat *unsatisfactory*. Why? It is too weak because we may be missing intensional functions that *do not have a term form* in our current definitional extension of set theory; and it is unsatisfactory because the axiom scheme, as tentatively stated above, *depends on our given choice of a definitional extension* of set theory. We can solve both problems in one blow by sticking to the basic language of set theory presented in §2. We then make the axiom scheme parametric on those set theory formulas $\varphi(x, y)$ with at most two free variables, x and y. The definitive version of the axiom scheme of replacement is then:

$$(Rep) \quad [(\forall x)(\exists! y)\ \varphi(x, y)] \ \Rightarrow\ [(\forall I)(\exists! T)(\forall u)\ (u \in T \ \Leftrightarrow\ (\exists i \in I)\ \varphi(i, u))].$$

Note that given a term $t(x)$, by choosing the formula $\varphi = (y = t(x))$ we obviously have $(Rep) \Rightarrow (Rep_0)$. However, in practice we frequently use (Rep) in its weaker (Rep_0) version.

Since, as shown in Exercise 120, the axiom of replacement provides a more general form of comprehension than the separation axiom, we can avail ourselves of a *more general notation* to define sets that implicitly uses both the replacement and separation axioms. Specifically, given any term $t(x)$ whose only variable is x (resp., a term $t(x_1, \ldots, x_n)$ with variables $x_1, \ldots, x_n$), a set theory formula $\varphi(x)$ (resp., $\varphi(x_1, \ldots, x_n)$), and a set A (resp., a set $W \subseteq A_1 \times \ldots \times A_n$), we can use the notation $\{t(x) \mid x \in A \ \wedge \varphi(x)\}$ (resp., $\{t(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in W \ \wedge \varphi(x_1, \ldots, x_n)\}$), to denote, by definition, the set

$$t[\{x \in A \mid \varphi(x)\}] \qquad (resp.,\ \ t[\{(x_1, \ldots, x_n) \in W \mid \varphi(x_1, \ldots, x_n)\}]).$$

For example, the set of sets of rational arrays of even length can be specified by the set expression $\{[n{\to}\mathbb{Q}] \mid n \in \mathbb{N} \wedge (\exists m)\ n = m + m\}$. Note that when writing a set expression like $t[\{(x_1, \ldots, x_n) \in W \mid \varphi(x_1, \ldots, x_n)\}]$ we are implicitly using the fact that, since $\mathscr{U}^n \subseteq \mathscr{U}$, the (Rep) axiom also applies to n-ary intensional functions. Specifically, we can always replace an n-ary function $\lambda(x_1, \ldots, x_n).\ t(x_1, \ldots, x_n)$ by a unary function extending it to the entire universe $\mathscr{U}$, such as, $\lambda x.$ **if** $x{:}\mathscr{U}^n$ **then** $t(p_1(x), \ldots, p_n(x))$ **else** $\varnothing$ **fi**.

[9] This intuition is confirmed by Theorem 11 in §13.2.

Exercise 120 *Prove that in ZF the axiom* (*Sep*) *of separation is not needed, since it is a consequence of* (*Rep*).

Exercise 121 *The axiom of replacement is sometimes stated in a seemingly stronger way as the axiom scheme that for any set theory formula $\varphi(x, y)$ with at most two free variables, x and y adds the axiom*

$$(Rep') \quad (\forall I)([(\forall i \in I)(\exists! y)\ \varphi(i, y)] \ \Rightarrow\ [(\exists! T)(\forall u)\ (u \in T \ \Leftrightarrow\ (\exists i \in I)\ \varphi(i, u))]).$$

Prove that, in the context of all other axioms of set theory, (*Rep*) *and* (*Rep′*) *are equivalent. (Hint: associate to each formula $\varphi(x, y)$ another formula $\varphi'(x, y)$ such that $(\forall x)(\exists! y)\ \varphi'(x, y)$ holds, and such that $\varphi'(x, y)$ behaves exactly like $\varphi(x, y)$ when restricted to any set I such that $(\forall i \in I)(\exists! y)\ \varphi(i, y)$ holds).*

Exercise 122 *(Generalizes both Exercise 34 and Theorem 11). Prove (using both Repl and AC) the following results for $f : A \longrightarrow B$ an I-indexed function:*

1. *If $f : A \longrightarrow B$ is a left inverse, then f is injective.*
2. *If $f : A \longrightarrow B$ is injective and $\bigtimes A \neq \emptyset$, then f is a left inverse.*
3. *$f : A \longrightarrow B$ is a right inverse iff f is surjective.*
4. *$f : A \longrightarrow B$ is both a left and a right inverse iff f is bijective.*

Exercise 123 *Prove that the model $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\})$ defined in §5.4 satisfies the* (*Rep*) *axiom.*

Chapter 17

Case Study: Dependent and Polymorphic Types in Maude

As already mentioned in §16.3.3, intensional functions, which at first sight might appear to be quite ethereal, abstract entities, are eminently practical and are used routinely in programming languages supporting dependent and polymorphic types. In particular, polymorphic types, also called *generic* or *parameterized* types, are not only supported by declarative languages such as ML, Haskell, and Maude, but also by imperative ones such as C++ and Java. There is also a strong interest at the software engineering level in *generic programming methodologies* to achieve very high levels of software reusability and quality (see, e.g., [22, 50]).

This case study is an exercise in mathematical modeling where, as in §5, we use equational logic to develop computable models of set-theoretic entities. In this case the computable models are not models of sets, but of intensional functions. Haven't we seen this before? Why, of course! Section 5.6 already presented a computable model in equational logic of hereditarily finite sets with atoms, which in hindsight can be recognized as an intensional function mapping each set of atoms with an equality predicate to the set of hereditarily finite sets built from such atoms. The emphasis here will be on concrete examples, as opposed to general foundations and correctness proofs. I will revisit foundational issues in §**??**. This case study exemplifies the power of mathematical modeling in *two* complementary directions: (1) we give computable models of intensional functions in equational logic; and (2) we give a *set-theoretic semantics* to dependent and polymorphic types as intensional functions, a theme further developed in §**??**.

17.1 Dependent Types in Maude

How are dependent types defined in Maude? As parameterized functional modules. The only difference between dependent and polymorphich types is that in a dependent type the parameter x ranges over the elements $x \in A$ of a *set* A, whereas in a polymorphic type it ranges over the sets $x : \mathscr{A}$ of a proper *class* $\mathscr{A}$. But how can a parameterization $x \in A$ be captured by a parameter theory? Yes, very easily. Since we are interested in computable models, we may assume that A is a space, and, as we did in §5.4 to characterize HF-sets, we may further assume that A has been characterized as the set of canonical forms associated to a specific sort, say `A`, in a functional module, say `fmod FOO is` $(\Sigma, E \cup B)$ `endfm`, satisfying requirements (1)–(5) in §5.3. Then we can define the following functional theory:

```
fth A* is protecting FOO .
  op x : -> A .
endfth
```

As already mentioned in §5.6, the meaning of the `protecting FOO` declaration is very strong: it means not only that `FOO` is imported, but also that its has a *fixed model interpretation*. Which model? What one might vaguely call the *standard* or *intended* model, and is more precisely called the *initial* model of `FOO` (I discuss initial models in detail in §**??**). For example, for the Maude modules `BOOL-OPS`, `NAT`, and `RAT` their initial models are, respectively, the Boolean algebra with two elements, the natural numbers, and the

rational numbers. Fortunately, since we assume that FOO satisfies requirements (1)–(5) in §5.3, we already *know* what this initial model is, even before getting to §**??**. Indeed, as explained in §5.3, if FOO is a theory $T = (\Sigma, E \cup B)$ satisfying requirements (1)–(5) and Ω is its constructor subsignature, then its canonical forms give us a model —which we shall see in §**??** is initial, and is called an order-sorted Σ-algebra— with data elements given by the family of sets $\{Can_{T,s}\}_{s \in S}$, with $Can_{T,s} = \{can(t) \in T_{\Omega/B} \mid t \in T_{\Sigma,s}\}$, and where for each function symbol $f : s_1 \ldots s_n \longrightarrow s$ in Σ we have an associated function f_T specified by the defining equality

$$f_T([t_1]_B, \ldots, [t_n]_B) = can(f(t'_1, \ldots, t'_n))$$

for $t'_1 \in [t_1]_B, \ldots, t'_n \in [t_1]_B$, such that $t'_1 \in T_{\Omega,s_1}, \ldots, t'_n \in T_{\Omega,s_n}$. For example, for $T =$ NAT and + the addition function symbol, $+_{\mathtt{NAT}}$ is just what one would expect: the addition function on natural numbers.

So, coming back to our theory A*, what are its models? Well, its data part is fixed and must be exactly the family of sets $\{Can_{\mathtt{FOO},s}\}_{s \in S}$ for FOO $= (\Sigma, E \cup B)$, and the operations $f_{\mathtt{FOO}}$ are likewise fixed. So the only choice possible is on how to interpret the constant x, which *must* be an element of the fixed set $Can_{\mathtt{FOO},A}$. Therefore, the models of A* are in essence the choices for the parameter $x \in Can_{\mathtt{FOO},A}$. *Dependent types* are then specified by parameterized functional modules whose parameter theory is of the above general form A*. Let us see some examples. Consider the theory

```
fth NzNat* is protecting NAT .
  op n : -> NzNat .
endfth
```

which describes the parameterization $n \in \mathbb{N}^+$, where, by definition, $\mathbb{N}^+ = \mathbb{N} - \{0\}$ is the set of nonzero natural numbers. We can then model the dependent type $\{\mathbb{N}_n\}_{n \in \mathbb{N}^+}$, where, by definition, $\mathbb{N}_n$ are the residue classes modulo n, by means of the following parameterized functional module:

```
fmod NAT/{n :: NzNat*} is
 sort Nat/{n} .
 op [_] : Nat -> Nat/{n} [ctor] .
 op _+_ : Nat/{n} Nat/{n} -> Nat/{n} .
 op -_ : Nat/{n} -> Nat/{n} .
 op _*_ : Nat/{n} Nat/{n} -> Nat/{n} .
 vars I J : Nat .
 ceq [I] = [I rem n] if I >= n .
 eq [I] + [J] = [I + J] .
 ceq -[I] = [sd(n,I)] if I < n .
 eq [I] * [J] = [I * J] .
endfm
```

where rem and sd are, respectively, the remainder and symmetric difference functions in the NAT module. This module defines for each $n \in \mathbb{N}^+$ the residue classes $\mathbb{N}_n$ modulo n (sometimes also denoted $\mathbb{Z}_n$), as well as their addition, minus, and multiplication functions. It is well-known that $\mathbb{N}_n$ with addition, minus and zero element [0] is a commutative group, and that residue class multiplication is associative and commutative, has [1] as its identity, and distributes over addition. That is, $\mathbb{N}_n$ is not just a set but a *commutative ring*, i.e., a Σ-algebra (see Def. 17) for the unsorted signature $\Sigma = \{(0,0), (1,0), (-,1), (+,2), (*,2)\}$, that satisfies all the above-mentioned equational properties. As we did for groups in §16.1.1, we can easily define the class **CRing** of commutative rings. Furthermore, in each instance of this module for the different choices of $n \in \mathbb{N}^+$, the canonical forms by the equations give us a computable model for $\mathbb{N}_n$ and all its operations. Therefore, the above parameterized functional module gives us a computable model for the intensional function $\mathbb{N}_{\_}$ with set-to-class typing:

$$\mathbb{N}_{\_} : \mathbb{N}^+ \longrightarrow \mathbf{CRing} : n \mapsto \mathbb{N}_n$$

which of course by the axiom of replacement has also a set-to-set typing. Let us instantiate our module with $n = 7$ (which since 7 is prime makes $\mathbb{N}_7$ not only a commutative ring but a *field*, so that there is a division operation for nonzero residue classes).

```
view 7 from NzNat* to NAT is op n to term 7 . endv

fmod NAT/7 is protecting NAT/{7} . endfm
```

The above view illustrates a feature not seen in earlier examples, namely, that in a view we can map a constant or a function symbol not just to another constant or function symbol, but to a *term*. Here 7 is not really a constant like 0 is, but abbreviates the term $s(s(s(s(s(s(s(0)))))))$ in decimal notation. Let us see some evaluations.

```
reduce in NAT/7 : [5] + [5] .
result Nat/{7}: [3]

reduce in NAT/7 : - [3] .
result Nat/{7}: [4]

reduce in NAT/7 : [3] + [4] .
result Nat/{7}: [0]

reduce in NAT/7 : [3] + - [6] .
result Nat/{7}: [4]

reduce in NAT/7 : [3] * [3] .
result Nat/{7}: [2]

reduce in NAT/7 : [3] * [5] .
result Nat/{7}: [1]

reduce in NAT/7 : [2] * [4] .
result Nat/{7}: [1]
```

Note that the last two evaluations illustrate the fact that $\mathbb{N}_7$ is a field.

Another dependent type we can consider is the type $\{\mathcal{P}(n)\}_{n\in\mathbb{N}^+}$, for which we can obtain a computable model using the following parameterized functional module:

```
fmod POW{n :: NzNat*} is
 sorts Pow{n} NatMagma FinNatSet .
 subsorts Nat < NatMagma .
 subsorts Pow{n} < FinNatSet .
 op {} : -> Pow{n} [ctor] .              *** empty set
 op _,_ : NatMagma NatMagma -> NatMagma [assoc comm ctor] .
 op {_} : NatMagma -> FinNatSet [ctor] .
 op _in_ : Nat NatMagma -> Bool .        *** membership in a magma
 op _C=_ : NatMagma NatMagma -> Bool .   *** magma containment
 op preds : NzNat -> NatMagma .          *** predecessors of a number
 op all : -> NatMagma .                  *** all predecessors of n

 vars I J : Nat .         vars M M' : NatMagma .

 eq M , M = M .                           *** idempotency
 eq preds(s(0)) = 0 .
 eq preds(s(s(I))) = s(I) , preds(s(I)) .
 eq all = preds(n) .
 eq I in J = I == J .
 eq I in J,M = if I == J then true else I in M fi .
 eq M C= M = true .
 eq M C= M,M' = true .
 eq I C= M = I in M .
 ceq I,M C= M' = false if I in M' = false .
 cmb {M} : Pow{n} if M C= all = true .
endfm
```

The sort structure of this module is similar to that of the `HF-SETS` module in §5.1, except that we use natural numbers as atoms and, since we are only interested in powersets of the form $\mathcal{P}(n)$, we do not need to iterate the pairing operator. Note that we have not only a parametric sort `Pow{n}`, whose canonical forms give us a computable model for $\mathcal{P}(n)$, but also a sort `FinNatSet`, whose canonical forms give us a model for the set $\mathcal{P}_{fin}(\mathbb{N})$ of finite subsets of the natural numbers. Note also that membership, resp. containment, are not defined between an element and a set, resp. two sets, but between an element and a magma, resp. two magmas, by functions `_in_` and `_C=_`. For convenience, the module makes use of the built-in equality predicate == for natural numbers and of a predefined `if_then_else_fi` function; both these functions could

be easily defined by explicit equations if desired. The elements of `Pow{n}` are either the empy set, denoted here by `{}`, or, as specified by the last conditional membership, are nonempty finite sets of natural numbers with all their elements smaller than n. In particular, for each $n \in \mathbb{N}^+$ the canonical forms of sort `Pow{n}` give us a computable model for the intensional function $\mathcal{P}$ with set-to-class typing:

$$\mathcal{P} : \mathbb{N}^+ \longrightarrow \mathscr{U}_{fin} : n \mapsto \mathcal{P}(n)$$

which, again by the axiom of replacement, has also a set-to-set typing. Let us see some evaluations, for $n = 7$, in the module: `fmod POW-7 is protecting POW{7} . endfm`.

```
reduce in POW-7 : {1,3,5} .
result Pow{7}: {1,3,5}

reduce in POW-7 : {1,3,7} .
result FinNatSet: {1,3,7}

reduce in POW-7 : all .
result NatMagma: 0,1,2,3,4,5,6

reduce in POW-7 : (1,3,5) C= 1,3,4,5 .
result Bool: true

reduce in POW-7 : (1,3,4) C= 1,4,5 .
result Bool: false
```

17.2 Polymorphic-and-Dependent Types in Maude

A parameter $x : \mathscr{A}$ of a *polymorphic type* ranges over a proper *class* $\mathscr{A}$. However, the same way, and for the same reasons, that intensional functions can have several arguments, parameterized functional modules can have several parameter theories, giving rise to different parameters. Some of these parameters may range over proper classes, whereas other parameters may range over the elements of a set. Therefore, in general we may have parameterized functional modules that define types which are *both* polymorphic in some parameters, and dependent in some other parameters.

Let us first consider a polymorphic type specified by a parameterized functional module with a single parameter ranging over the proper class **Atom** defined by the equivalence:

$$x{:}\mathbf{Atom} \;\Leftrightarrow\; (\exists A, f)\; x = (A, f) \;\wedge\; f \in [A{\to}2] \;\wedge\; ((\forall a, a' \in A)\; f(a, a') = 1 \;\Leftrightarrow\; a = a').$$

This class was axiomatized by the parameter theory `ATOM` in §5.6. The parameterized functional module `HF-SETS{X :: ATOM}` then specifies a computable model for the following intensional function, which we denote $\mathscr{V}_\omega(\_)$, with class-to-class typing:

$$\mathscr{V}_\omega(\_) : \mathbf{Atom} \longrightarrow \mathscr{U} : (A, f) \mapsto \mathscr{V}_\omega(A, f)$$

where if A is an equationally-defined space of atoms and f is an equationally-defined computable equality predicate, then $\mathscr{V}_\omega(A, f)$ is the computable model of canonical forms of sort `Set{A}` obtained by instantiating `HF-SETS{X :: ATOM}` with a view `A` mapping the `ATOM` parameter theory to (A, f). In §5.6 we saw such instances for (A, f) instantiated to natural numbers with their equality predicate, and to quoted identifiers, again with their equality predicate. Of course, the case of HF-sets without atoms is just the instance $\mathscr{V}_\omega(\varnothing, \varnothing) = \mathscr{V}_\omega$, which was discussed in detail in §5.1 and §5.4. One could object that a set A of atoms is not supposed to belong to the the class $\mathscr{U}$ of all sets. In a sense this objection is legitimate, since the whole point of having atoms is to have elements which are not sets and therefore are not in $\mathscr{U}$. But since we know that such a set of atoms could in principle be modeled by a set in $\mathscr{U}$, we can just assume that $A : \mathscr{U}$ for mathematical modeling purposes. In another sense, however, the above objection is based on a too naive notion of "atom". Why should sets in $\mathscr{U}$ be excluded *a priori* from being used as atoms? What is an atom anyway? Whatever we *decide* to make into an atom. How is something made into an atom? By *encapsulation*, that is, by *ignoring* its *internal structure*. How is this technically accomplished in the $\mathscr{V}_\omega(A, f)$ construction? Through the `[_]` operator that encapsulates atoms into set elements and achieves

a complete separation between atoms and sets. But notice than an atom *can*, and often does have an internal structure. For example, the set $\{[1],[2],[3]\}$ is really the set $\{[s(0)],[s(s(0))],[s(s(s(0)))]\}$ when we represent numbers as term in Peano notation, or the set $\{[\{\emptyset\}],[\{\{\emptyset\}\}],[\{\{\{\emptyset\}\}\}]\}$ when we represent them as sets in their Zermelo representation. The point is that the internal structure of those atoms is completely *irrelevant* for their use *qua atoms*; so they may just as well be sets.

How about mixed types that are *both* dependent and polymorphic? We can develop computable models for them (when this is possible) by using parameterized functional modules which have several parameters, some ranging over elements of a set and others over proper classes. Here is one, which defines simultaneously the following three intensional functions[1]: (i) $\lambda(A,f):\mathbf{Atom}.\ \mathbb{N}^+\times A$, (ii) $\lambda(A,f):\mathbf{Atom}.\ \mathcal{P}_{fin}(\mathbb{N}^+\times A)$, and (iii) $\lambda(n,(A,f)):\mathbb{N}^+\times\mathbf{Atom}.\ [n{\to}A]$, where in (iii), by convention, we have identified $\mathbb{N}^+$ with the set-defining class of its elements, which can be easily specified as a formula using the axiom of infinity.

```
fmod ARRAY{n :: NzNat*,A :: ATOM} is protecting NAT/{n} .
  sorts  BagNat PairNat{A}  MagmaPairNat{A} FinRelNat{A} Fun{n,A} .
  subsort Nat < BagNat .                  subsort PairNat{A} < MagmaPairNat{A} .
  subsort Fun{n,A} < FinRelNat{A} .
  op _ _ :  BagNat BagNat ->  BagNat [ctor assoc comm] .            *** bag union
  op [_,_] : Nat A$Atom -> PairNat{A} [ctor] .                      *** ordered pair ctor
  op _,_ :  MagmaPairNat{A}  MagmaPairNat{A} ->  MagmaPairNat{A} [ctor assoc comm] .
  op {} : -> FinRelNat{A} [ctor] .                                 *** empty relation
  op {_} : MagmaPairNat{A} -> FinRelNat{A} [ctor] .
  op _in_ :  PairNat{A}  MagmaPairNat{A} -> Bool .                 *** membership in magma
  op preds :  NzNat -> BagNat .                                    *** predecessors
  op all : -> BagNat .                                                    *** all preds
  op dom :  MagmaPairNat{A} -> BagNat [memo] .                            *** domain
  op _ _ : Fun{n,A} Nat/{n} -> A$Atom .                                    *** function eval

  vars I J : Nat .   var A A' : A$Atom .   var M M' : MagmaPairNat{A} .  var B : BagNat .

  eq M , M = M .                                                   *** idempotency
  eq [I,A] in [J,A'] = (I == J) and equal(A,A') .
  eq [I,A] in [J,A'] , M = ((I == J) and equal(A,A')) or [I,A] in M .
  eq preds(s(0)) = 0 .
  eq preds(s(s(I))) = s(I) preds(s(I)) .
  eq all = preds(n) .
  eq dom([I,A]) = I .
  eq dom([I,A],M) = if [I,A] in M then dom(M) else I dom(M) fi .
  cmb {M} : Fun{n,A} if dom(M) = all .
  ceq {[I,A],M}[I] = A  if dom([I,A],M) = all .
endfm
```

In this module: (i) the canonical forms for the parameterized sort `PairNat{A}` provide a computable model for the intensional function $\lambda(A,f):\mathbf{Atom}.\ \mathbb{N}^+\times A$, (ii) the canonical forms for the parameterized sort `FinRelNat{A}` provide a computable model for the intensional function $\lambda(A,f):\mathbf{Atom}.\ \mathcal{P}_{fin}(\mathbb{N}^+\times A)$, and (iii) the canonical forms for the parameterized sort `Fun{n,A}` provide a computable model for the intensional function $\lambda(n,(A,f)):\mathbb{N}^+\times\mathbf{Atom}.\ [n{\to}A]$, which is just the function composition

$$\mathbb{N}^+\times\mathbf{Atom}\hookrightarrow\mathscr{U}^2\xrightarrow{[\_\to\_]}\mathscr{U}$$

where the first arrow denotes class inclusion, and were, as already mentioned, we have identified $\mathbb{N}^+$ with the set-defining class of its elements. Function (iii) is a good example of a mixed data type, which is polymorphic on the variable (A,f), and dependent on the variable n. The specification is quite similar to the one for `POW{n :: NzNat*}`, but a few remarks may be helpful. The sort `BagNat` provides not magmas, but *bags* (also called multisets) of natural numbers. And the domain of a finite relation R of sort `FinRelNat{A}` is: (i) not defined for the relation itself, but for its underlying magma, and (ii) results not on a set or magma, but in a bag of sort `BagNat`. This has the important advantage of recording the *multiplicity* with which the same number n may appear as first component of several pairs in R. The key insight then is to realize that a function $f\in[n{\to}A]$ is exactly a relation R whose domain consists of all predecessors of n *without repetitions*, i.e., of the bag `all`. This is captured by the conditional membership in the module.

[1]Properly speaking, these intensional functions are partial. However, as discussed in Exercise 115, they can be easily extended to total ones.

Function application (defined by the last equation) is interesting. It could of course be defined as a partial function (using a typing at the kind level) whose second argument could be any natural number. But it is conceptually more pleasing to take advantage of the previous module `NAT/{n :: NzNat*}` (which, up to the slight change of representation between $[k]$ and k, gives us a model for the finite type of predecessors of n) to obtain function evaluation as a *total* function. As a final remark, the `memo` attribute for the `dom` function has no mathematical significance but is useful computationally: it instructs Maude to *memoize* the result of the computation, so that it will take zero time thereafter. This is useful to keep applying a function repeatedly to many arguments, without having to pay ever again the computational cost of checking that it is a function. Let us instantiate this module to form arrays of rational numbers and see some evaluations.

```
view Rat-eq from ATOM to RAT is
 sort Atom to Rat .
 op equal to _==_ .
endv

fmod ARRAY-7-RAT is protecting ARRAY{7,Rat-eq} . endfm

reduce in ARRAY-7-RAT : all .
result BagNat: 0 1 2 3 4 5 6

reduce in ARRAY-7-RAT : {[1,1/3],[2,1/4],[2,1/5]} .
result FinRelNat{Rat-eq}: {[1,1/3],[2,1/4],[2,1/5]}

reduce in ARRAY-7-RAT : dom([1,1/3],[2,1/4],[2,1/5]) .
result BagNat: 1 2 2

reduce in ARRAY-7-RAT : {[0,0],[1,1],[2,1/2],[3,1/3],[4,1/4],[5,1/5],[6,1/6]} .
result Fun{7,Rat-eq}: {[0,0],[1,1],[2,1/2],[3,1/3],[4,1/4],[5,1/5],[6,1/6]}

reduce in ARRAY-7-RAT : {[0,0],[1,1],[2,1/2],[3,1/3],[4,1/4],[5,1/5],[6,1/6]} [4] .
result PosRat: 1/4

reduce in ARRAY-7-RAT : {[0,0],[1,1],[2,1/2],[2,1/7],[3,1/3],[4,1/4],[5,1/5],[6,1/6]} [4] .
result [Rat,BagNat]: {[0,0],[1,1],[2,1/2],[2,1/7],[3,1/3],[4,1/4],[5,1/5],[6,1/6]} [4]
```

The last evaluation is interesting. The last conditional equation in the module blocks any nonfunctional relation from being applied to an argument. In this case, since 2 appears twice in the relation's domain, the equation's condition fails, and the expression is returned as an error term of *kind* `[Rat,BagNat]`. Maude uses the set of maximal sorts in a connected component to name the (implicitly defined) kind above them. Here, since we have subsort inclusions `Nat < Rat` and `Nat < Bagnat`, the relevant kind of errors for function applications is precisely `[Rat,BagNat]`.

17.3 Definability Issues

All the examples of intensional functions discussed in this case study correspond to intensional functions that are *definable* by terms in some definitional extension of set theory. Or do they? Well, using the axiom of infinity and a few primitive recursive functions one can certainly define the naturals modulo n as an intensional function (this is an interesting exercise). And all other intensional functions use well-known set-forming operators such as formation of ordered pairs, powersets or finite power sets, and function spaces, except one. Which one? the function $\lambda(A, f) : \mathbf{Atom}.\ \mathscr{V}_\omega(A, f)$. It is not obvious that this is an intensional function. Using (a straightforward generalizarion of) Exercise 27, we could write it in the somewhat more palatable form $\lambda(A, f) : \mathbf{Atom}.\ \bigcup_{n\in\mathbb{N}} \mathcal{P}^n_{fin}(A)$, but even in that form it is not obvious that this is an intensional function. Why not? because the notation $\mathcal{P}^n_{fin}(A)$ conveniently hides an *inductive definition*: $\mathcal{P}^0_{fin}(A) = A$, and $\mathcal{P}^{n+1}_{fin}(A) = \mathcal{P}_{fin}(\mathcal{P}^n_{fin}(A))$, which it is not obvious we can describe as an intensional function. That it can be so described requires a discussion of *transfinite constructions*, and a proof showing that such constructions are definable as intensional functions. This is done in §18.4. So, in the end the claim that all the dependent, polymorphic, and mixed dependent and polymorphic types in this case study are definable as intensional functions will be vindicated; but some more work needs to be done.

Chapter 18

Well Orders, Ordinals, Cardinals, and Transfinite Constructions

Well-ordered sets are well-founded chains and are intimately connected with the axiom of choice. Ordinals provide canonical representatives of all the isomorphism classes of well-ordered sets, and have very useful inductive and recursive properties that make possible many transfinite set-theoretic constructions, including that of cardinals.

18.1 Well-Ordered Sets

Well-ordered sets were first studied by Cantor as part of his theory of ordinals (see [9] §12–13). In particular, Theorem 21 is Cantor's Theorem N in [9] §13.

Definition 29 *A* well-ordered set *is a strict poset* $(A, >)$ *such that it is a chain and* $(A, >)$ *is well-founded.*

Of course, by duality, $(A, >)$ is a strict poset and a chain iff $(A, <)$ is a strict poset and a chain. However, for well-ordered sets it is very convenient to use the notation $>$ with the usual meaning of "greater than," since then $(A, >)$ being well-founded means that there are no infinite descending sequences $a_0 > a_1 > \ldots > a_n > \ldots$ Each natural number n is well-ordered by $>$ the usual descending order; in particular, for $n = 0$, this means that the empty chain $(\varnothing, \varnothing)$ is well-ordered. Similarly, $(\mathbb{N}, >)$, $(\mathbb{N} \cup \{\infty\}, >)$, with $\infty > n$ for each $n \in \mathbb{N}$, and $(\mathbb{N} \times \mathbb{N}, Lex(>, >))$ are all well-ordered. Note that if $(A, >)$ is well-ordered and $B \subseteq A$, then $(B, >|_B)$ is also well-ordered. For example, the sets of odd numbers, of even numbers, and of prime numbers, are all well-ordered by $>$, because $(\mathbb{N}, >)$ is well-ordered. In particular, if $(A, >)$ is well-ordered and $a \in A$, then $(>[\{a\}], >|_{>[\{a\}]})$, called the *initial segment*, or *subchain*, of $(A, >)$ *below* a, is also well-ordered. This yields yet another way to see that $(n, >)$ is well-ordered for any $n \in \mathbb{N}$, since the initial segments of $(\mathbb{N}, >)$ are precisely the well-orderings $(n, >) = (>[\{n\}], >|_{>[\{n\}]})$ for each $n \in \mathbb{N}$ (as we shall soon see, the equalities $n = (>[\{n\}])$ for each $n \in \mathbb{N}$ show that all natural numbers and also $\mathbb{N}$ itself are *ordinals*). To simplify notation, initial segments $(>[\{a\}], >|_{>[\{a\}]})$ will sometimes be abbreviated to $(>[\{a\}], >)$, leaving implicit the restriction for $>$.

If $(A, >)$ is well-ordered, since $>$ is well-founded, every non-empty subset $B \subseteq A$ has a $>|_B$-minimal element. But since $(A, >)$ is a chain this exactly means that: (i) every non-empty subset $B \subseteq A$ has an inf (for the $<$ order), $\bigwedge B$, and (ii) $\bigwedge B \in B$. In particular, if $A \neq \varnothing$, then $(A, >)$ has a bottom element $\bot = \bigwedge A$. Note that any well-ordered $(A, >)$ is *almost* a complete lower semilattice, since the only inf that may be missing is $\bigwedge \varnothing$. In the case of $(\mathbb{N} \cup \{\infty\}, >)$, we indeed have a complete semilattice with $\bigwedge \varnothing = \infty$, and therefore a complete lattice; likewise, any $(s(n), >)$ is trivially a complete lattice. Note, furthermore, that if $(A, >)$ is well-ordered, then the function $\bigwedge_{\_} : \mathcal{P}(A) - \{\varnothing\} \longrightarrow A : B \mapsto \bigwedge B$ is a *choice function*. There is, therefore, an intimate connection between well-orders and the axiom of choice that is further explored in §18.5.

Exercise 124 *Prove that a well-founded set* (A, R) *is well-ordered iff for each* $x, y \in A$ *we have:* $x\,R\,y \;\vee\; y\,R\,x \;\vee\; x = y$.

A very nice property about well-ordered sets is that they are all *comparable*, in the sense that given any two well-ordered sets $(A, >)$ and $(B, >)$, either they are relation-isomorphic (therefore poset-isomorphic), or one of them is isomorphic to an initial segment of the other. To show this result we need a few lemmas.

Lemma 20 *Let $(A, >)$ be a well-ordered set. Then there is no $a \in A$ such that we have a poset isomorphism $(A, >) \cong (>[\{a\}], >)$.*

Proof. We reason by contradiction. Suppose that there is an $a \in A$ for which we have a relation isomorphism $h : (A, >) \longrightarrow (>[\{a\}], >)$. Then $a \in \{x \in A \mid x \neq h(x)\}$. Let $a_0 = \bigwedge\{x \in A \mid x \neq h(x)\}$. We cannot have $h(a_0) < a_0$, since then we would have $h(a_0) = h(h(a_0))$, contradicting the injectivity of h. Therefore, $h(a_0) > a_0$; but then, since h is strictly monotonic, and is the identity on $>[\{a_0\}]$, we must have $a_0 \notin h[A]$, contradicting the fact that, since $h(a_0) > a_0$, $h[A] = (>[\{a\}])$, and h is strictly monotonic, $a_0 \in h[A]$. □

Corollary 6 *Let $(A, >)$ and $(B, >')$ be well-ordered sets, and let $a \in A$, $b, b' \in B$, be such that $(>[\{a\}], >) \cong (>'[\{b\}], >')$, and $(>[\{a\}], >) \cong (>'[\{b'\}], >')$. Then $b = b'$.*

Proof. By composing isomorphisms we have $(>'[\{b\}], >') \cong (>'[\{b'\}], >')$. If $b \neq b'$, without loss of generality we may assume $b > b'$. Therefore, $(>'[\{b\}], >')$ is isomorphic to its initial segment $(>'[\{b'\}], >')$, contradicting Lemma 20. □

Lemma 21 *Let $f, g : (A, >) \longrightarrow (B, >')$ be two relation isomorphisms between two well-ordered sets. Then $f = g$, that is, relation isomorphisms between well-ordered sets, if they exist, are unique.*

Proof. Suppose $f \neq g$ and let $a = \bigwedge\{x \in A \mid f(x) \neq g(x)\}$. Without loss of generality we may assume $f(a) > g(a)$. Since f and g are strictly monotonic we must have $f(a), g(a) \notin f[>[\{a\}]] = g[>[\{a\}]]$. But then, by f strictly monotonic, $g(a) \notin f[A]$, contradicting $f[A] = B$. □

Lemma 22 *Let $(A, >)$ and $(B, >')$ be well-ordered sets, and let $a \in A$, $b \in B$, be such that there is a relation isomorphism $h : (>[\{a\}], >) \longrightarrow (>'[\{b\}], >')$. Then for each $a' \in (>[\{a\}])$, h restricts to a relation isomorphism $h : (>[\{a'\}], >) \longrightarrow (>'[\{h(a')\}], >')$.*

Proof. It is enough to show that $h[>[\{a'\}]] = (>'[\{h(a')\}])$. By h strictly monotonic we have $h[>[\{a'\}]] \subseteq (>'[\{h(a')\}])$. Towards a contradiction, since $>'[\{h(a')\}] \subset (>'[\{b\}]) = h[>[\{a\}]]$, assume that there is $a'' \in (>[\{a\}])$ with $h(a'') \in (>'[\{h(a')\}] - h[>[\{a'\}]])$. Then $h(a') > h(a'')$, and therefore, $a' > a''$. Therefore, $h(a'') \in h[>[\{a'\}]]$, contradicting $h(a'') \in (>'[\{h(a')\}] - h[>[\{a'\}]])$. □

We are now ready for the main theorem on comparability of well-ordered sets.

Theorem 21 *(Cantor) Let $(A, >)$ and $(B, >')$ be well-ordered sets. Then, either $(A, >) \cong (B, >')$, or there exists $a \in A$ such that $(>[\{a\}], >) \cong (B, >')$, or there exists $b \in B$ such that $(A, >) \cong (>'[\{b\}], >')$.*

Proof. Suppose $(A, >) \ncong (B, >')$, and let $h = \{(a, b) \in A \times B \mid (>[\{a\}], >) \cong (>'[\{b\}], >')\}$. By Corollary 6, h is a partial function from A to B. Let A_0 be the domain of h, and $B_0 = h[A_0]$. Again, by Corollary 6, $h : A_0 \longrightarrow B_0$ is injective and therefore bijective. If $A_0 \neq A$, let $a = \bigwedge(A - A_0)$. Obviously, $>[\{a\}] \subseteq A_0$, and, since $a \notin A_0$, $\nexists b \in B$ such that $(>[\{a\}], >) \cong (>'[\{b\}], >')$. I claim that $>[\{a\}] = A_0$. Suppose not and let $a_0 \in A_0 - (>[\{a\}])$; then we must have $a_0 > a$. Therefore, $a \in (>[\{a_0\}])$. But since $(>[\{a_0\}], >) \cong (>[\{h(a_0)\}], >')$, by Lemma 22 there is a $b \in (>[\{h(a_0)\}])$ such that $(>[\{a\}], >) \cong (>[\{b\}], >')$, contradicting the fact that $\nexists b \in B$ with $(>[\{a\}], >) \cong (>'[\{b\}], >')$. Since $h^{-1} = \{(b, a) \in B \times A \mid (>[\{a\}], >) \cong (>'[\{b\}], >')\}$, the same argument proves that if $B_0 \neq B$, then $>'[\{b\}] = B_0$, where $b = \bigwedge(B - B_0)$.

Next, let us prove that the bijection $h : A_0 \longrightarrow B_0$ is a relation isomorphism $h : (A_0, >) \longrightarrow (B_0, >')$. By Exercise 60 it is enough to show that h is a relation homomorphism. Let $a, a' \in A_0$ with $a > a'$. By construction we have $(>[\{a\}], >) \cong (>[\{h(a)\}], >')$, and by Lemma 22, there is a $b' \in (>[\{h(a)\}])$ such that $(>[\{a'\}], >) \cong (>[\{b'\}], >')$. Therefore, by Corollary 6, $h(a') = b' \in (>[\{h(a)\}])$, and we have $h(a) > h(a')$, as desired.

Since we are assuming $(A, >) \ncong (B, >')$, if $A = A_0$, then we must have $>'[\{b\}] = B_0$, where $b = \bigwedge(B - B_0)$. Symmetrically, if $B = B_0$, then we must have $>[\{a\}] = A_0$, where $a = \bigwedge(A - A_0)$. The case $>[\{a\}] = A_0$ and $>'[\{b\}] = B_0$ is impossible, since it would give us the contradiction $(a, b) \in h$. □

18.2 Ordinals

Cantor thought of an ordinal as the abstract concept associated to a class of isomorphic well-ordered sets. But such a class is not a set. In the 1920's John von Neumann made the crucial contribution of defining an ordinal as a unique representative of such an isomorphism class (see [52], 346–354).

By Theorem 21 and Lemma 22, the class **WOrd** of all well-ordered sets can be endowed with a *strict order* by the defining equivalence

$$(B, >') > (A, >) \quad \Leftrightarrow \quad (\exists b \in B)\ (A, >) \cong (>[\{b\}], >').$$

Of course, if we could choose a canonical representative, let us call it $ord(A, >)$, for the class of all well-ordered sets $(A', >')$ such that $(A, >) \cong (A', >')$, then, by Theorem 21, the subclass, let us call it **Ord**, determined by those canonical representatives would become a *chain* $(\mathbf{Ord}, >)$ (of course, a chain structure on a class that is not a set, let us call it a "megachain"). Ordinals are precisely those canonical representatives; therefore we call $ord(A, >) \cong (A, >)$ the *ordinal* of $(A, >)$. They were defined and studied by John von Neumann. As we shall see in §18.5.1, ordinals solve in one blow the pending issue of explicitly defining a canonical representative $|A|$, that is, a *cardinal*, for the class of all sets in bijection with a set A.

Recall, from Exercise 64, that for any chain, and therefore for any well-ordered set $(A, >)$, the function $>[\{\_\}] = \lambda a \in A.\ >[\{a\}] \in \mathcal{P}(A)$ defines a poset isomorphism $>[\{\_\}] : (A, >) \longrightarrow (>[\{\_\}][A], \supset)$ inside the powerset poset $(\mathcal{P}(A), \supset)$.

Definition 30 *A well-ordered set* $(A, >)$ *is called an* ordinal *iff the poset isomorphism* $>[\{\_\}]$ *is the* identity function id_A*, that is, iff for each* $a \in A$ *we have* $a = (>[\{\ a\}])$.

This definition of ordinals looks a bit remarkable, but, as already mentioned, it is for example satisfied by each natural number $(n, >)$, and therefore also by $(\mathbb{N}, >)$, that is, each natural number, and the set of natural numbers, are ordinals. Note, by the way, the interesting fact that $(n, >) = (n, \supset) = (n, \ni)$, and $(\mathbb{N}, >) = (\mathbb{N}, \supset) = (\mathbb{N}, \ni)$. Of course, if $(A, >)$ is an ordinal, we always have $(A, >) = (A, \supset)$; we show below that for any ordinal we also have $(A, \supset) = (A, \ni)$.

Since the above definition of ordinals is of course definable by a set-theory formula with a single variable, ordinals form a class **Ord**, and we have a subclass inclusion $\mathbf{Ord} \subset \mathbf{WOrd}$. As customary, we shall use greek letters $\alpha, \beta, \gamma, \delta, \ldots$ to denote ordinals. However, the natural numbers $n \in \mathbb{N}$ are still denoted as usual, but the letter ω is used to refer to poset $(\mathbb{N}, >)$. We adopt the helpful *abuse of notation* of systematically conflating α with $(\alpha, \supset)$, which is reasonable, since the order $\supset$ is entirely determined by the set α, because it is always the relation $\supset_\alpha = \{(x, y) \in \alpha \times \alpha \mid x \supset y\}$, which we sometimes abbreviate to just $\supset$. This notation makes it unnecessary to write $\alpha : \mathbf{Ord}$ each time, since the notation α already *assumes* $\alpha : \mathbf{Ord}$.

Lemma 23 *If* $(\alpha, \supset_\alpha)$ *is an ordinal, then* $(\alpha, \supset_\alpha) = (\alpha, \ni_\alpha)$*, where, by definition,* $\ni_\alpha = \{(x, y) \in \alpha \times \alpha \mid x \ni y\}$.

Proof. Let $x, y \in \alpha$. Then $x \supset y$ implies $y \in (\supset_\alpha[\{\ x\}]) = x$, and therefore $x \ni y$. Conversely, $x \ni y$ implies $y \in (\supset_\alpha[\{\ x\}])$, and therefore $x \supset y$. □

Lemma 24 *If* $(\alpha, \supset_\alpha)$ *is an ordinal and* $x \in \alpha$*, then* $(x, \supset_x)$ *is also an ordinal.*

Proof. Let $x \in \alpha$. Since $x = (\supset_\alpha[\{\ x\}])$, we have $x \subset \alpha$, and therefore $(x, \supset_x)$ is well-ordered. We just need to show that for each $y \in x$ we have $y = (\supset_x[\{\ y\}])$. But since $x \subset \alpha$, we have $y \in \alpha$, and therefore $y = (\supset_\alpha[\{\ y\}])$; so we only need to show $(\supset_x[\{\ y\}]) = (\supset_\alpha[\{\ y\}])$. Indeed, since $y \in x$ implies $y \in (\supset_\alpha[\{\ x\}])$, and therefore $x \supset y$, we have $(\supset_x[\{\ y\}]) = \{z \in (\supset_\alpha[\{\ x\}]) \mid y \supset z\} = (\supset_\alpha[\{\ y\}])$, as desired. □

Lemma 25 *Let* α *be an ordinal, and let* $\alpha \supset X$*. Then* $(X, \supset_X)$ *is an ordinal iff* $X \in \alpha$.

Proof. The $(\Leftarrow)$ implication follows from Lemma 24. To see the $(\Rightarrow)$ implication, let $\beta = \bigwedge(\alpha - X)$. By the minimality of β, for each $x \in \beta$ we have $\beta \supset x$, and therefore $x \in X$, so that $\beta \subseteq X$. We will be done if we show $X \subseteq \beta$. Let $\gamma \in X$; if $\gamma \supset \beta$, then $\beta \in \gamma \subset X$, which is impossible since $\beta \in (\alpha - X)$. Therefore, $\gamma \subseteq \beta$, but since $\gamma \in X$, we have $\gamma \neq \beta$. Therefore, $\gamma \subset \beta$, and therefore $\gamma \in \beta$, as desired. □

Lemma 26 *Let* α, β *be ordinals. Then* $\alpha \cap \beta$ *is an ordinal.*

Proof. For each $\gamma \in \alpha \cap \beta$ we have $\gamma = (\supset_\alpha[\{\ \gamma\}]) = (\supset_\beta[\{\ \gamma\}])$. Therefore, $\gamma = (\supset_\alpha[\{\ \gamma\}]) \cap (\supset_\beta[\{\ \gamma\}]) = (\supset_{\alpha\cap\gamma}[\{\ \gamma\}])$, as desired. □

Theorem 22 *For any two ordinals, α, β, either $\alpha \subseteq \beta$ or $\beta \subseteq \alpha$. As a consequence, $\alpha \cup \beta$ is an ordinal.*

Proof. Suppose not. Then $\alpha \cap \beta \subset \alpha$ and $\alpha \cap \beta \subset \beta$. But by Lemma 26, $\alpha \cap \beta$ is an ordinal, and by Lemma 25, $\alpha \cap \beta \in \alpha$ and $\alpha \cap \beta \in \beta$. Therefore, $\alpha \cap \beta \in \alpha \cap \beta$. Therefore, $\alpha \cap \beta \subset \alpha \cap \beta$, which is impossible. □

Corollary 7 *For any two ordinals, α, β, $(\alpha, \supset) \cong (\beta, \supset)$ iff $\alpha = \beta$.*

Proof. The implication ($\Leftarrow$) is trivial. To see the implication ($\Rightarrow$), assume $(\alpha, \supset) \cong (\beta, \supset)$ with $\alpha \neq \beta$. By Theorem 22, without loss of generality we may assume $\alpha \subset \beta$. Therefore, by Lemma 25 $\alpha \in \beta$. But since $(\alpha, \supset) = (\supset[\{\ \alpha\}], \supset)$, this means that we have an isomorphism $(\supset[\{\ \alpha\}], \supset) \cong (\beta, \supset)$, which is impossible by Lemma 20. □

It follows trivially from Theorem 22 and Corollary 7 that when we restrict to the subclass **Ord** the relation $(B, >') > (A, >)$ on well-orders, we obtain exactly the relation $\alpha \supset \beta$. Furthermore, Theorem 22 shows that $(\mathbf{Ord}, \supset)$ is a chain (a "megachain," since we shall see in §18.2.1 that **Ord** is a proper class). Also, note that, by Theorem 7, for any well-ordered set $(A, >)$ such that $(A, >) \cong (\alpha, \supset)$, and $(A, >) \cong (\beta, \supset)$, we must have $\alpha = \beta$. Therefore, ordinals provide a unique, canonical representative for each class of isomorphic well-ordered sets containing an ordinal. To see that **Ord** provides canonical representatives for *all* well orders, we still need to show that any well-ordered set is isomorphic to an ordinal. This will follow as an easy corollary of the following theorem.

Theorem 23 *Let $(A, >)$ be such that for each $a \in A$, the well-order $(>[\{a\}], >)$ is isomorphic to an ordinal. Then $(A, >)$ itself is also isomorphic to an ordinal.*

Proof. Consider the formula

$$(x \notin A \Rightarrow y = \emptyset) \ \wedge \ (x \in A \Rightarrow y{:}\mathbf{Ord} \ \wedge \ y \cong (>[\{x\}], >).$$

By Corollary 7, this formula defines an intensional function o which can be typed as $o : A \longrightarrow \mathbf{Ord}$, and therefore, by the axiom of replacement, a surjective extensional function $o : A \twoheadrightarrow O$, where $O = \{o(x) \mid x \in A\}$. Since all the elements of O are ordinals, by Theorem 22 the poset $(O, \supset)$ is a chain. To see that $o : (A, >) \twoheadrightarrow (O, \supset)$ is a poset isomorphism it is enough, by Exercise 60, to show that o is strictly monotonic. But if $x, y \in A$ with $x > y$, then $(>[\{x\}], >) > (>[\{y\}], >)$, and since $(>[\{x\}], >) \cong o(x)$, and $(>[\{y\}], >) \cong o(y)$, this forces $o(x) \supset o(y)$, as desired. We need to show that $(O, \supset)$ is an ordinal. That is, that for each $o(x)$ we have $o(x) = (\supset_O[\{o(x)\}])$. We have an isomorphism $h_x : (>[\{x\}], >) \longrightarrow (o(x), \supset)$ by hypothesis, which for each $y < x$ restricts to an isomorphism $h_x : (>[\{y\}], >) \longrightarrow (\supset[\{h_x(y)\}], \supset)$. Therefore, by Corollary 7, we have $o(y) = (\supset[\{h_x(y)\}]) = h_x(y)$. But since o is an isomorphism we have, $\supset_O[\{o(x)\}] = \{o(y) \mid x > y\} = \{h_x(y) \mid x > y\} = o(x)$, as desired. □

Corollary 8 *Each well-ordered set is isomorphic to a unique ordinal.*

Proof. Uniqueness follows from Corollary 7. By Theorem 23, it is enough to show that if $(A, >)$ is a well-ordered set, then for each $a \in A$ $(>[\{a\}], >)$ is isomorphic to an ordinal. Towards a contradiction, suppose that $A \neq \{x \in A \mid (\exists y)\ y{:}\mathbf{Ord} \ \wedge \ (>[\{x\}], >) \cong y\}$, and let $a = \bigwedge(A - \{x \in A \mid (\exists y)\ y{:}\mathbf{Ord} \ \wedge \ (>[\{a\}], >) \cong y\})$. Therefore, $>[\{a\}] \subseteq \{x \in A \mid (\exists y)\ y{:}\mathbf{Ord} \ \wedge \ (>[\{x\}], >) \cong y\}$, and therefore, by Theorem 23, $(>[\{a\}], >)$ is isomorphic to an ordinal, contradicting a's definition. □

It follows directly from Corollary 8 that the formula $(\neg(x : \mathbf{WOrd}) \Rightarrow y = \emptyset) \wedge (x : \mathbf{WOrd} \Rightarrow (y : \mathbf{Ord} \ \wedge \ x \cong y))$, where here the predicate $\cong$ denotes *poset isomorphism*, defines an intensional function *ord* which has a typing

$$ord : \mathbf{WOrd} \longrightarrow \mathbf{Ord}$$

and is such that for any well-ordered sets $(A, >)$ and $(B, >')$:

- $(A, >) \cong ord(A, >)$,
- $(A, >) > (B, >')$ iff $ord(A, >) \supset ord(B, >')$, and
- $(A, >) \cong (B, >')$ iff $ord(A, >) = ord(B, >')$.

Therefore, ordinals classify, and capture the essence of, all the different well orders. Waxing Platonic, one could say that they provide the *ideal patterns* for understanding all well-orders.

18.2.1 Ordinals as Transitive Sets

The notion of transitive set provides an alternative, very useful characterization of ordinals.

Definition 31 *A set A is* transitive *iff* $(\forall a \in A)\ a \subseteq A$.

It follows immediately from the definition of ordinal (and was explicitly pointed out in the proof of Lemma 24), that any ordinal is a transitive set. Why on earth are transitive sets called "transitive"? The answer lies in their following "transitive-like," equivalent characterizations (1)–(2) below:

Exercise 125 *Prove the following:*

1. *A set A is transitive iff* $(\forall x \in A)\ y \in x \Rightarrow y \in A$.
2. *A set A is transitive iff for each $n \in \mathbb{N}$ and each sequence of sets $x_0, \ldots, x_n$ such that $A \ni x_0 \ni \ldots \ni x_n$ we have $x_0, \ldots, x_n \in A$.*
3. *A set is transitive iff* $\bigcup A \subseteq A$.
4. *A set is transitive iff* $A \subset \mathcal{P}(A)$.
5. *If a set A is transitive, then $\mathcal{P}(A)$ is also transitive.*
6. *If $\{A_i\}_{i \in I}$ is an I-indexed family of transitive sets, then both $\bigcup_{i \in I} A_i$ and $\bigcap_{i \in I} A_i$ are transitive.*
7. *Given a set A, the membership relation $\ni_A = \{(x, y) \in A \times A \mid x \ni y\}$ is transitive iff every $x \in A$ is a transitive set.*

Note that, in the light of Exercise 125–(7), Lemma 24 gives us an independent confirmation for something we already knew because of Lemma 23, namely, that for any ordinal α the membership relation $\ni_\alpha$ is transitive. How can ordinals be characterized as transitive sets?

Theorem 24 *A set A is an ordinal iff A is transitive and $(A, \ni_A)$ is a well-ordered set.*

Proof. The ($\Rightarrow$) implication follows from ordinals being transitive and Lemma 23. For the ($\Leftarrow$) implication, let A be transitive and with $(A, \ni_A)$ well-ordered. By Lemma 23 we just need to show that for each $a \in A$ we have $a = (\ni_A[\{a\}])$. But since A is transitive, we have $a \subseteq A$ and therefore $a = \{x \in A \mid a \ni x\} = \{x \in A \mid a \ni_A x\} = (\ni_A[\{a\}])$, as desired. □

The above characterization of ordinals makes it very easy to prove that **Ord** is a proper class. Indeed, the proof of this result shows that assuming that there is a set $\mathbb{O}$ of all ordinals immediately leads to a Russell-like paradox, the so-called Burali-Forti paradox, which was discovered in 1897 by Cesare Burali-Forti (see [52], 104–112).

Theorem 25 *(Burali-Forti). The class* **Ord** *is proper.*

Proof. Assume not, so that there is a set $\mathbb{O}$ such that $x \in \mathbb{O} \Leftrightarrow x : \mathbf{Ord}$. Then for each $\alpha \in \mathbb{O}$, if $\beta \in \alpha$, β is an ordinal, so that $\beta \in \mathbb{O}$. Therefore, $\mathbb{O}$ is a transitive set. Also, $(\mathbb{O}, \ni_{\mathbb{O}})$ is obviously a chain, and it is well-ordered, since we cannot have an infinite descending chain of ordinals $\alpha_0 \ni \alpha_1 \ni \ldots \ni \alpha_n \ni \alpha_{n+1} \ni \ldots$ Therefore, $\mathbb{O}$ is an ordinal, and as a consequence, $\mathbb{O} \in \mathbb{O}$. But this means that $\mathbb{O} \subset \mathbb{O}$, and therefore that $\mathbb{O} \neq \mathbb{O}$, a vicious contradiction. □

18.2.2 Successor and Limit Ordinals

The transitive set characterization of ordinals makes it easier to both study and construct the two possible kinds of ordinals there are, namely, limit ordinals and successor ordinals.

Definition 32 *An ordinal α is called a* limit ordinal[1] *iff* $(\forall \beta \in \alpha)\ (\exists \gamma)\ \alpha \ni \gamma \ni \beta$. *An ordinal that is not a limit ordinal is called a* successor ordinal.

[1]The ordinal 0 is typically excluded from the definition of limit ordinal, so that we then have a trychotomy of an ordinal being either 0, or a limit ordinal, or a sucessor ordinal. I adopt a broader definition of limit ordinal that holds vacuously for 0 and also for the ordinals that are limits of all ordinals below them in a proper sense.

Note that both 0 and ω are limit ordinals. Note also that, by definition, α is a successor ordinal iff there exists $\beta \in \alpha$ such that there is no γ such that $\alpha \ni \gamma \ni \beta$. We call such a β (which is unique because α is a chain) the *predecessor* of α and use the notation $p(\alpha) = \beta$. Note that if α is a successor ordinal, since α is a chain and $p(\alpha)$ is a maximal element, $p(\alpha)$ is indeed the *top* element of the chain α (in the inverse, $\subset$ order, and a bottom element in the $\supset$ order). Therefore, successor ordinals can be equivalently characterized as *those ordinals having a top element* as posets in ascending order; and of course limit ordinals are exactly the *topless* ordinals. Note that each natural number of the form $s(n)$ is a successor ordinal with $p(n) = n$.

Lemma 27 *The intensional successor function $s = \lambda x.\ x \cup \{x\}$ has a typing $s : \mathbf{Ord} \longrightarrow \mathbf{Ord}$. Furthermore, for each ordinal α, $s(\alpha)$ is a successor ordinal.*

Proof. We need to show for each ordinal α that $s(\alpha)$ is an ordinal and is a successor ordinal. But if α is transitive and well-ordered by $\ni_\alpha$, then $s(\alpha)$ just adds the new element α on top of the chain α and is well-ordered by $\ni_{s(\alpha)}$, and is transitive, since if $x \in \alpha \cup \{\alpha\}$, either $x \in \alpha$ and then $x \subseteq \alpha \subset \alpha \cup \{\alpha\}$, or $x = \alpha$ and then $\alpha \subset \alpha \cup \{\alpha\} = s(\alpha)$. Of course, since α is the top element of $s(\alpha)$, the ordinal $s(\alpha)$ is a successor ordinal. □

As already discussed, the class **Ord** with the order $(\mathbf{Ord}, \supset)$ is a (descending) "megachain" by Lemma 22, and, dually, also an (ascending) "megachain" with the order $(\mathbf{Ord}, \subset)$. Since **Ord** is a proper class, it has subclasses that are also proper. It is easy to see that if $\mathscr{A} \subseteq \mathbf{Ord}$ is a proper class, then there is no sup $\bigvee \mathscr{A}$ in $\mathscr{A}$, since there cannot be any upper bound α for $\mathscr{A}$ (otherwise, we would have an inclusion $\mathscr{A} \subseteq \alpha$, against the assumption that $\mathscr{A}$ was proper). Since a proper subclass $\mathscr{A} \subseteq \mathbf{Ord}$ cannot have an upper bound, *a fortiori* it cannot have a least upper bound $\bigvee \mathscr{A}$ in the megachain $(\mathbf{Ord}, \subset)$. However, if A is a *set* and $A \subseteq \mathbf{Ord}$, then, first of all $(A, \supset_A)$ is a well-ordered set, since it is a chain and we cannot have an infinite descending chain

$$\alpha_0 \supset \alpha_1 \supset \ldots \alpha_n \supset \alpha_{n+1} \supset \ldots$$

in A, since this would automatically violate the well-foundedness of the ordinal α_0. Furthermore, the least upper bound $\bigvee A$ always exists in the "megachain" $(\mathbf{Ord}, \subset)$. As we shall see, this is very useful to construct limit ordinals.

Theorem 26 *The intensional function $\bigcup = \lambda x.\ \bigcup x$ has a typing $\bigcup : \mathcal{S}(\mathbf{Ord}) \longrightarrow \mathbf{Ord}$. Furthermore, this function is a sup operator for all subsets in the "megachain" $(\mathbf{Ord}, \subset)$.*

Proof. For each subset $A \subseteq \mathbf{Ord}$, we have to show two things: (i) that $\bigcup A : \mathbf{Ord}$, and (ii) that $\bigcup A = \bigvee A$ in the "megachain" $(\mathbf{Ord}, \subset)$. To prove (i) we need to show that $\bigcup A$ is transitive and well-ordered by $\ni$. But $\bigcup A$ is transitive by Exercise 125–(6), and it is well-ordered by $\ni$ because, since all the elements of an ordinal are also ordinals, we have $\bigcup A \subset \mathbf{Ord}$, and therefore $(\bigcup A, \supset_A) = (\bigcup A, \ni_A)$ is a well-founded set. Therefore, $\bigcup A : \mathbf{Ord}$. To prove (ii), just note that any upper bound of A under inclusion must necessarily contain $\bigcup A$ as a subset. □

Note that the two operations s and $\bigcup$ allow us to generate all ordinals. Specifically,

Theorem 27 *For each ordinal α:*

1. *α is a limit ordinal iff $\alpha = \bigcup \alpha$.*

2. *α is a successor ordinal iff there exists an ordinal β such that $\alpha = s(\beta)$.*

Proof. Note that, since α is transitive, by Exercise 125–(3), we always have $\bigcup \alpha \subseteq \alpha$. Therefore, (1) is equivalent to proving that α is a successor ordinal iff $\bigcup \alpha \subset \alpha$. But since $\bigcup \alpha$ is an ordinal, we have $\bigcup \alpha \subset \alpha$ iff $\bigcup \alpha \in \alpha$, which makes $\bigcup \alpha$ the top element of α; and if α has a top element, then this top element contains all elements of α and is therefore their union $\bigcup \alpha$. For (2), the ($\Leftarrow$) part was already shown in Theorem 27. To see the ($\Rightarrow$) part, let α be a successor ordinal and let β be its top element. Then $x \in \alpha$ iff $x = \beta \ \vee \ x \in \beta$, therefore, $\alpha = \beta \cup \{\beta\}$, as desired. □

18.2.3 Ordinal Arithmetic

Ordinal arithmetic goes back to Cantor, who defined all the operations and identified many of its laws (see [9], §14 and ff.). Since we shall show in §18.5.1 that cardinals are a subclass of ordinals, and for cardinals κ, λ we have already defined addition and multiplication operations $\kappa + \lambda$ and $\kappa \cdot \lambda$ in §15.1, to avoid any confusions of notation between operations on cardinals and operations on ordinals, I will use $\alpha \boxplus \beta$ for ordinal addition, and $\alpha \otimes \beta$ for ordinal multiplication. This choice of notation is not without some residual ambiguities, since $A \boxplus B$ was used in §4.2 to denote the symmetric difference of two sets, and $A \otimes B$ was used in §4.3 to denote the set of unordered pairs a, b with $a \in A$ and $b \in B$. However, these residual notational clashes seem so remote as to cause no harm. The *standard* notation for ordinal addition and multiplication in many set theory textbooks is $\alpha + \beta$ and $\alpha \cdot \beta$. However, since cardinal arithmetic has much better algebraic properties than ordinal arithmetic, and therefore is the *right* transfinite generalization of natural number arithmetic, I think it better to reserve the standard notation for cardinal arithmetic.

The operations $\alpha \boxplus \beta$ and $\alpha \otimes \beta$ are just the canonizations of two corresponding operations on any two well-ordered sets $(A, >)$ and $(B, >')$, namely, the operations $(A, >) \oplus (B, >')$, and $(B \times A, Lex(>', >))$ (note the "twist" between A and B in the second construction). That is, we *define* $\alpha \boxplus \beta$ and $\alpha \otimes \beta$ by the defining equalities

$$\alpha \boxplus \beta = ord((\alpha, \ni_\alpha) \oplus (\beta, \ni_\beta)) \qquad \alpha \otimes \beta = ord(\beta \times \alpha, Lex(\ni_\beta, \ni_\alpha)).$$

We are already familiar with the operation $(B \times A, Lex(>', >))$, which was defined for posets in Exercise 62, and for well-founded relations in §14.1.1; and since it is easy to check (exercise) that the lexicographic combination of two chains is a chain, if $(A, >)$ and $(B, >')$ are well-ordered, the poset $(B \times A, Lex(>', >))$ is well-ordered. The construction $(A, >) \oplus (B, >')$ on two well-ordered sets consists on "putting the chain $(B, >')$ on top of the chain $(A, >)$." Of course, we first need to make the two chains disjoint. Therefore, $(A, >) \oplus (B, >') = (A \oplus B, \oplus(>, >'))$, where $A \oplus B$ is the usual disjoint union of sets, and where the order $\oplus(>, >')$ is defined by cases as follows:

1. $(\forall a, a' \in A)\ \ (a, 0) \oplus (>, >')\ (a', 0) \ \Leftrightarrow\ a > a'$
2. $(\forall b, b' \in B)\ \ (b, 1) \oplus (>, >')\ (b', 1) \ \Leftrightarrow\ b >' b'$
3. $(\forall a \in A)(\forall b \in B)\ \ (b, 1) \oplus (>, >')\ (a, 0)$.

It is then easy to check (exercise) that if $(A, >)$ and $(B, >')$ are well-ordered, then $(A, >) \oplus (B, >')$ is well-ordered.

Exercise 126 *The notation $(A, >) \oplus (B, >')$ might fool us into believing that we have an analogue for well-ordered sets of the property in Exercise 39 for disjoint unions; but this is* false*. Give an example of two monotonic functions $f : (A, >) \longrightarrow (C, >'')$ and $g : (B, >') \longrightarrow (C, >'')$, such that the function $[f, g] : A \oplus B \longrightarrow C$ is not a monotonic function $[f, g] : (A, >) \oplus (B, >) \longrightarrow (C, >'')$. However, $[f, g] : A \oplus B \longrightarrow C$* is *monotonic if we drop clause (3) in the definition of $\oplus(>, >')$; but then we obtain a poset that, as soon as A or B are nonempty, is* not *a chain. This suggest several sweeping generalizations. Prove that, by dropping clause (3), we can get an analogue of the property in Exercise 39 for: (i) posets and monotonic functions; (ii) posets and strictly monotonic functions; (iii) well-founded relations and relation homomorphisms; and (iv) binary relations on sets and relation homomorphisms. (Hint: You can cut your work fourfold by realizing that (iv) implies (i)–(iii)). (For category theory aficionados: in the language of category theory, this shows that the categories of: (i) posets and monotonic functions; (ii) posets and strictly monotonic functions; (iii) well-founded relations and relation homomorphisms; and (iv) relations and relation homomorphisms all have* coproducts*).*

Exercise 127 *Prove in detail that the constructions $\alpha \boxplus \beta$ and $\alpha \otimes \beta$ are intensional functions $\_ \boxplus \_ : \mathbf{Ord} \times \mathbf{Ord} \longrightarrow \mathbf{Ord}$, and $\_ \otimes \_ : \mathbf{Ord} \times \mathbf{Ord} \longrightarrow \mathbf{Ord}$. (Hint: You can make your life easier by first showing that the constructions $(A, >) \oplus (B, >')$ and $(B \times A, Lex(>', >))$ are binary intensional functions on* **WOrd***, and then using the intensional function ord* : **WOrd** $\longrightarrow$ **Ord***.)*

Note that, since $\alpha \boxplus 1$ just adds a "fresh" top element to α, even when α is a successor ordinal and has already one, it is trivial to prove that for any ordinal α we have the identity $\alpha \boxplus 1 = s(\alpha)$. Therefore, the well-ordered set $(\mathbb{N} \cup \{\infty\}, >)$, with $\infty > n$ for each $n \in \mathbb{N}$, is isomorphic to $s(\omega) = \omega \boxplus 1$. Note also that, although for *finite* ordinals, ordinal addition and multiplication coincide with natural number addition and multiplication (this is obvious just by cardinality), in general the operations $\alpha \boxplus \beta$ and $\alpha \otimes \beta$ are *not* commutative. Indeed, we obviously have $1 \boxplus \omega = \omega \neq \omega \boxplus 1 = s(\omega)$. Similarly, $2 \otimes \omega = \omega \neq \omega \boxplus \omega = \omega \otimes 2$.

Furthermore, distributivity of multiplication on the right also fails, since we have $(1 \boxplus 1) \otimes \omega = 2 \otimes \omega = \omega \neq \omega \boxplus \omega = (1 \otimes \omega) \boxplus (1 \otimes \omega)$. Therefore, the algebraic properties of ordinal addition and multiplication are rather poor at the transfinite level. However they do enjoy *some* properties; for example:

Exercise 128 *Prove the following properties of ordinal addition:*

1. $\alpha \boxplus 0 = 0 \boxplus \alpha = \alpha$
2. $(\alpha \boxplus \beta) \boxplus \gamma = \alpha \boxplus (\beta \boxplus \gamma)$
3. $\alpha \boxplus s(\beta) = s(\alpha \boxplus \beta)$
4. *If β is a limit ordinal, then $\alpha \boxplus \beta = \bigcup\{\alpha \boxplus \nu \mid \nu \in \beta\}$.*

Exercise 129 *Prove the following properties of ordinal multiplication:*

1. $\alpha \otimes 0 = 0$
2. $\alpha \otimes 1 = \alpha$
3. $(\alpha \otimes \beta) \otimes \gamma = \alpha \otimes (\beta \otimes \gamma)$
4. $\alpha \otimes s(\beta) = (\alpha \otimes \beta) \boxplus \alpha$
5. $\alpha \otimes (\beta \boxplus \gamma) = (\alpha \otimes \beta) \boxplus (\alpha \otimes \gamma)$
6. *If β is a limit ordinal, then $\alpha \otimes \beta = \bigcup\{\alpha \otimes \nu \mid \nu \in \beta\}$.*

Note that it follows from Exercises 128 and 129 that $\omega \boxplus \omega = \omega \otimes 2$ is the first limit ordinal above ω, the next is $\omega \boxplus \omega \boxplus \omega = \omega \otimes 3$, and so on. In general, the limit ordinals are all of the form $\omega \otimes \alpha$ for some α; but using ordinal multiplication and Exercise 129-(6) we can "fast forward" in this megachain and get limit ordinals of limit ordinals, and so on. This gives us a better understanding of the "megachain" **Ord**, which we can picture as a transfinite ascending sequence of the form:
$0 < 1 < 2 < \ldots < n < \ldots < \omega <$
$\omega \boxplus 1 < \omega \boxplus 2 < \ldots < \omega \boxplus n < \ldots < \omega \otimes 2 <$
$\omega \otimes 2 \boxplus 1 < \omega \otimes 2 \boxplus 2 < \ldots < \omega \otimes 2 \boxplus n < \ldots < \omega \otimes 3 < \ldots$
$\omega \otimes 4 < \ldots < \omega \otimes 5 < \ldots < \omega \otimes n < \ldots < \omega \otimes \omega < \ldots$
$\omega \otimes \omega \otimes \omega < \ldots < \omega \otimes \omega \otimes \omega \otimes \omega < \ldots \omega \otimes \overset{n}{\ldots} \otimes \omega < \ldots$

All this might look like an arcane piece of set-theoretic *esoterica*, yet, quite remarkably, it is eminently *practical* for computer science applications. There are several reasons for this: (i) we musnt't forget that ordinals are well orders, and in particular well-founded relations, so they can be very useful in *termination proofs* for programs;[2] (ii) using yet another ordinal operation, namely, ordinal *exponentiation* (see Exercise 131, and [15] for a more detailed presentation of ordinal arithmetic, including exponentiation), it is possible to give a unique *symbolic* representation of ordinals as polynomial algebraic expressions (the so-called *Cantor normal form* of an ordinal) that for ordinals smaller than the so-called ϵ_0-ordinal do form a *computable data type* (see, e.g., [45] §11.8–9, and [35]); (iii) all this is exploited in theorem provers such as ACL2, Coq, PVS, HOL, Isabelle, and Mizar. In particular, in ACL2 ordinal arithmetic is mechanized and it is possible to use ordinals to prove the termination of functional programs in pure Lisp [35].

18.3 Transfinite Induction

Since **Ord** is a proper class, the "megachain" $(\mathbf{Ord}, \supset)$ is not itself a well-ordered *set*; but it is a "well-ordered class" in the sense that we cannot have an infinite descending sequence of ordinals

$$\alpha_0 \supset \alpha_1 \supset \ldots \alpha_n \supset \alpha_{n+1} \supset \ldots$$

since this would automatically violate the well-foundedness of the ordinal α_0, and in the even stronger sense that each nonempty *subclass* $\mathscr{A} \subseteq \mathbf{Ord}$ has an inf $\bigwedge \mathscr{A}$ such that $\bigwedge \mathscr{A} : \mathscr{A}$ in the "megachain" $(\mathbf{Ord}, \subset)$.

[2]Often, such termination proofs involve using a function that *decreases* as the computation advances. The most obvious range for functions of this kind is the well-ordered set ω of natural numbers, but this does not always work; the next thing often tried is a lexicographic ordering such as $\omega \otimes \omega$, but sometimes this may not be good enough either. The great thing about ordinals is that they give us an extremely useful *menagerie* of well-orders, since indeed we have seen that *any* well-order is isomorphic to some ordinal. Nevertheless, ordinals, since they are total orders, do not exhaust our repertoire of possible domains with which to prove termination: we may use *any* well-founded relation, which need not be a chain, and need not even be a partial order. For a survey of the use of well-founded relations in termination proofs of term rewriting systems see, e.g., [51] §6.

Theorem 28 *Each nonempty* subclass $\mathscr{A} \subseteq \mathbf{Ord}$ *has an inf* $\bigwedge \mathscr{A}$ *in the "megachain"* $(\mathbf{Ord}, \subset)$*, and furthermore* $\bigwedge \mathscr{A} : \mathscr{A}$.

Proof. Suppose that $\mathscr{A} \subseteq \mathbf{Ord}$ is nonempty, so there is an ordinal α such that $\alpha : \mathscr{A}$. Then we have $\bigwedge \mathscr{A} = \bigwedge\{\beta \in s(\alpha) \mid \beta \subseteq \alpha \wedge \beta : \mathscr{A}\}$, which exists and is such that $\bigwedge \mathscr{A} \in \{\beta \in s(\alpha) \mid \beta \subseteq \alpha \wedge \beta : \mathscr{A}\}$ by α well-ordered by $\supset$. Therefore, $\bigwedge \mathscr{A} : \mathscr{A}$. □

Technically, Theorem 28 is really a "theorem scheme," that is, a scheme that for each set theory formula $\mathscr{A}$ with a single variable x yields a set theory theorem. This theorem scheme then yields the following principle of *transfinite induction*.

Theorem 29 *(Transfinite Induction). Let* $\mathscr{A} \subseteq \mathbf{Ord}$ *be a subclass such that for each ordinal* α*,* $((\forall \gamma \in \alpha)\ \gamma : \mathscr{A}) \Rightarrow \alpha : \mathscr{A}$*. Then* $\mathscr{A} \equiv \mathbf{Ord}$.

Proof. Suppose that $\mathbf{Ord} - \mathscr{A}$ is nonempty, and let $\alpha = \bigwedge(\mathbf{Ord} - \mathscr{A})$. Then $\alpha \subseteq \mathscr{A}$, and therefore $\alpha : \mathscr{A}$, contradicting $\alpha : (\mathbf{Ord} - \mathscr{A})$. □

In applying the principle of transfinite induction to prove that a certain property $\mathscr{A}$ holds of all ordinals, one typically uses the distinction between successor ordinals and limit ordinals to prove that for each ordinal α one has $((\forall \gamma \in \alpha)\ \gamma : \mathscr{A}) \Rightarrow \alpha : \mathscr{A}$. That is, if α is a successor ordinal one can assume $\gamma : \mathscr{A}$ for $\gamma = p(\alpha)$ and for each $\gamma \in p(\alpha)$ to prove $\alpha : \mathscr{A}$; and if α is a limit ordinal one can likewise assume $\gamma : \mathscr{A}$ for each $\gamma \in \alpha$ to prove $\alpha : \mathscr{A}$. We will make ample use of transfinite induction in the following sections.

18.4 Transfinite Recursion

Consider the sequence of sets

$$\varnothing,\ \mathcal{P}(\varnothing),\ \mathcal{P}(\mathcal{P}(\varnothing)),\ \ldots, \mathcal{P}^n(\varnothing),\ \ldots$$

There is in principle no obvious set containing all such iterated powers, so to construct the union $\bigcup_{n\in\omega} \mathcal{P}^n(\varnothing)$ we first of all need an intensional function $\lambda n.\ \mathcal{P}^n(\varnothing)$, and then we need to use the axiom of replacement. However, the status and existence of the intensional function $\lambda n.\ \mathcal{P}^n(\varnothing)$, although intuitively a very reasonable function, since it is just a different notation for the above sequence, is not entirely obvious. Indeed, the notation $\lambda n.\ \mathcal{P}^n(\varnothing)$ is just an *abbreviation*, but for what? In the case of a function such as $\lambda x.\ s(x)$ there is a *term*, namely $s(x)$, so there is no doubt that such an intensional function exists. But $\mathcal{P}^n(\varnothing)$ looks like a term, but is not really a term; it just abbreviates a *different term* for each n, namely, the term $\mathcal{P}(\ldots(\mathcal{P}(\varnothing)\ldots)$ with n applications of the powerset operator to $\varnothing$. So, what is this function? It is a *recursive intensional function*. Indeed, the function $f = \lambda n.\ \mathcal{P}^n(\varnothing)$ has the recursive definition

- $f(0) = \varnothing$
- $f(s(n)) = \mathcal{P}(f(n))$.

But at present we only have a theory for *extensional* recursive functions, so we need to clarify the meaning and existence of such recursive *intensional* functions.

Furthermore, in some set theory constructions we often need to consider recursively-defined "sequences" of sets indexed not just by natural numbers, but by some infinite ordinal α beyond ω, so in general we may want to consider recursive intensional functions of the form $f : \alpha \longrightarrow \mathscr{U}$, so that the notion of "sequence" is now extended beyond ω in a transfinite way. This makes intuitively complete sense, since, after all, α is a well-ordered set and therefore a well-founded set, so we should be able to define recursive intensional functions from such a well-founded set α. Even more daringly, we sometimes need to consider transfinite recursively-defined "sequences" that are indexed not over the elements of a given ordinal α, but over *all ordinals*, that is, recursive intensional functions of the form $f : \mathbf{Ord} \longrightarrow \mathscr{U}$. This also makes intuitive sense, since, after all, **Ord** is a "well-ordered class," so we should be able to define such an f on a given α in terms of its values for cardinals smaller than α.

What should a theory of recursive intensional functions look like? Well, if we are going to define *intensional* functions, the defining task is unavoidably involved with *syntax*, since at the end of the day we must exhibit a set theory *formula* that defines such a function. In particular, the extensional notion of *step function* introduced in §14.3.2 needs to be replaced by an intensional version. Recall that, as explained in

Exercise 104, given a well-founded relation (A, R), to define a recursive function $f : A \longrightarrow B$, we can use a step function $h : [A \rightharpoonup B] \times A \longrightarrow B$.

But what are we going to use as a step function to define a recursive function $f : \mathbf{Ord} \longrightarrow \mathscr{U}$? There is no obvious analogue for the set $[A \rightharpoonup B]$, since the notation $[\mathbf{Ord} \rightharpoonup \mathscr{U}]$ does not describe a class.[3] However, the right intuition is that at any given transfinite stage α we will have defined, by the axiom of replacement, an *extensional* function $f\restriction_\alpha$, and we can then use it to define $f(\alpha) = h(f\restriction_\alpha, \alpha)$. Although in general there is no *set* that will contain all such extensional functions $f\restriction_\alpha$ for all ordinals α, there is obviously a class that will, namely $\mathscr{U}$, so we can adopt as our notion of step function an intensional function

$$h : \mathscr{U} \times \mathbf{Ord} \longrightarrow \mathscr{U}.$$

Of course, for pairs (x, α) where x is not a function, or where x is a function but has the wrong domain, the value of $h(x, \alpha)$ will be useless; but h can assign to such useless pairs (x, α) a similarly useless value like $\varnothing$, so there is no harm in the extra generality provided by the universe $\mathscr{U}$ (we could of course use the more restrictive class **Fun** of functions, and consider instead step functions of the form $h : \mathbf{Fun} \times \mathbf{Ord} \longrightarrow \mathscr{U}$).

To better ground our intuition, let us consider an example of such an h. What step function h should we use to define the recursive extensional function $\lambda n.\ \mathcal{P}^n(\varnothing)$? We could define $h : \mathscr{U} \times \mathbf{Ord} \longrightarrow \mathscr{U}$ by the lambda expression

$$\lambda(g, n).\ \mathbf{if}\ (n \in \omega - \{0\} \ \wedge\ g{:}\mathbf{Fun} \ \wedge\ dom(g) = n)\ \mathbf{then}\ \mathcal{P}(g(p(n))\ \mathbf{else}\ \varnothing\ \mathbf{fi}.$$

Indeed, this intensional step function gives the correct recursive definition for $f = \lambda n.\ \mathcal{P}^n(\varnothing)$, since $f(0) = h(\emptyset, 0) = \varnothing$, and $f(s(n)) = h(f\restriction_{s(n)}, s(n)) = \mathcal{P}(f(n))$.

We will proceed from the less to the more general, that is, we will first define recursive intensional functions of the form $f : \alpha \longrightarrow \mathscr{U}$. I will call such functions *α-recursive functions*; of course, by the axiom of replacement, any intensional α-recursive function has an associated extensional function $f : \alpha \twoheadrightarrow f[\alpha]$. Along the way we will obtain an intensional version of the simple recursion theorem for $\mathbb{N}$ as a special case of α-recursion. We will then treat the general case of recursive intensional functions of the form $f : \mathbf{Ord} \longrightarrow \mathscr{U}$, which can be called *transfinitely recursive functions*.

18.4.1 α-Recursion

Theorem 30 *(α-Recursion). Let $h : \mathscr{U} \times \mathbf{Ord} \longrightarrow \mathscr{U}$ be an intensional function, and let α be an ordinal. Then there exists an intensional function $rec(h)_\alpha$, with typing $rec(h)_\alpha : \alpha \longrightarrow \mathscr{U}$ such that*

$$(\forall \beta \in \alpha)\ rec(h)_\alpha(\beta) = h(rec(h)_\alpha\restriction_\beta, \beta).$$

Furthermore, $rec(h)_\alpha$ is extensionally unique*, in the sense that for any other intensional function $f : \alpha \longrightarrow \mathscr{U}$ such that $(\forall \beta \in \alpha)\ f(\beta) = h(f\restriction_\beta, \beta)$ the extensional functions $f : \alpha \twoheadrightarrow f[\alpha]$ and $rec(h)_\alpha : \alpha \twoheadrightarrow rec(h)_\alpha[\alpha]$ are identical.*

Proof. Extensional uniqueness follows from the following, more general lemma:

Lemma 28 *Let f, g be extensional functions such that $\beta = dom(f) = dom(g) \in s(\alpha)$ and such that $(\forall \gamma \in \beta)\ f(\gamma) = h(f\restriction_\gamma, \gamma) \ \wedge\ g(\gamma) = h(g\restriction_\gamma, \gamma)$. Then $f = g$.*

Proof. Suppose not, and let $\mu = \bigwedge\{\gamma \in \beta \mid f(\gamma) \neq g(\gamma)\}$. By the minimality of μ we then have $f\restriction_\mu = g\restriction_\mu$, and therefore $f(\mu) = h(f\restriction_\mu, \mu) = h(g\restriction_\mu, \mu) = g(\mu)$, contradicting $f(\mu) \neq g(\mu)$. □

Existence of the *extensional* function $rec(h)_\alpha : \alpha \twoheadrightarrow rec(h)_\alpha[\alpha]$ is equivalent to the truth of the formula $(\exists! f)\ A(f) \ \wedge\ dom(f) = \alpha$, where the predicate $A(f)$ holds of functions, called *approximations*, that satisfy the recursive definition and have domain $\beta \in s(\alpha)$. That is, we define the unary predicate A by the equivalence

$$A(f) \ \Leftrightarrow\ f{:}\mathbf{Fun} \ \wedge\ dom(f) \in s(\alpha) \ \wedge\ (\forall \gamma \in dom(f))\ f(\gamma) = h(f\restriction_\gamma, \gamma).$$

[3] It does describe *something*, namely, the set of equivalence classes under $\equiv$ of formulas $\varphi(x, y)$ in the language of set theory such that: (i) $ZFC \vdash (\forall x, y, y')\ (\varphi(x, y) \wedge \varphi(x, y')) \Rightarrow y = y'$, and (ii) $ZFC \vdash (\forall x, y)\ \varphi(x, y) \Rightarrow x : \mathbf{Ord}$. But such a quotient set of formulas is obviously *not* a class in set theory.

Existence of the *intensional* function $rec(h)_\alpha : \alpha \longrightarrow \mathscr{U}$ follows then easily from that of the extensional function $rec(h)_\alpha : \alpha \twoheadrightarrow rec(h)_\alpha[\alpha]$, since we can define the intensional function $rec(h)_\alpha$ by the equivalence

$$y = rec(h)_\alpha(x) \;\Leftrightarrow\; (x \notin \alpha \Rightarrow y = \varnothing) \;\wedge\; (x \in \alpha \Rightarrow ((\exists! f)\; A(f) \;\wedge\; dom(f) = \alpha \;\wedge y = f(x))).$$

Therefore, all boils down to proving $(\exists! f)\; A(f) \wedge dom(f) = \alpha$, which by Lemma 28 is equivalent to proving $(\exists f)\; A(f) \wedge dom(f) = \alpha$, which, in turn, is equivalent to proving $\alpha \in \{\beta \in s(\alpha) \mid (\exists f)\; A(f) \wedge dom(f) = \beta\}$. Suppose not, and let $\mu = \bigwedge\{\beta \in s(\alpha) \mid \neg((\exists f)\; A(f) \wedge dom(f) = \beta)\}$. By the minimality of μ, $(\gamma \in \mu) \Rightarrow \gamma \in \{\beta \in s(\alpha) \mid (\exists f)\; A(f) \wedge dom(f) = \beta\} \Rightarrow (\exists f)\; A(f) \wedge dom(f) = \gamma$. This, together with Lemma 28, means that we can prove $(\forall x)(\exists! y)\; \varphi$ for φ the formula

$$(x \notin \mu \Rightarrow y = \varnothing) \;\wedge\; (x \in \mu \Rightarrow (\exists f)\; A(f) \;\wedge\; dom(f) = x \;\wedge\; y = f).$$

Therefore, φ defines an intensional function *approx* with typing $approx : \mu \longrightarrow \mathbf{Fun}$, where for each $\gamma \in \mu$, $approx(\gamma)$ is the unique f such that $A(f) \;\wedge\; dom(f) = \gamma$.

Consider now the set of functions $approx[\mu]$. For any $\gamma, \gamma' \in \mu$, if $\gamma \in \gamma'$, then, by Lemma 28, we must have $approx(\gamma) = approx(\gamma')\restriction_\gamma$, and therefore $approx(\gamma) \subset approx(\gamma')$. This means that $(approx[\mu], \subset)$ is a *chain*. But it is trivial to prove (exercise) that the union of any ascending chain of functions (under inclusion) is itself a function. Therefore, $f_0 = \bigcup approx[\mu]$ is a function with $dom(f_0) = \bigcup \mu$; therefore, for each $\gamma \in \mu$ we have $f_0\restriction_\gamma = approx(\gamma)$. But then, if μ is a limit ordinal this means that $A(f_0) \wedge dom(f_0) = \mu$, contradicting $\mu \in \{\beta \in s(\alpha) \mid \neg((\exists f)\; A(f) \;\wedge\; dom(f) = \beta)\}$. And if μ is a successor ordinal, say $\mu = s(\nu)$, we can extend $f_0 = approx(\nu)$ to $f_0' = f_0 \cup \{(\nu, h(\nu, f_0))\}$, so that $A(f_0') \wedge dom(f_0') = \mu$, contradicting again $\mu \in \{\beta \in s(\alpha) \mid \neg((\exists f)\; A(f) \;\wedge\; dom(f) = \beta)\}$. □

18.4.2 Simple Intensional Recursion

The α-recursion theorem yields as an easy corollary an intensional version of the simple recursion theorem for $\mathbb{N}$, whose extensional version was proved in Theorem 5. Simple intensional recursion is used all the time, almost invisibly, in many set-theoretic constructions, so it is very much worthwhile to make its foundations explicit. As a matter of fact, our example recursive intensional function $\lambda n.\; \mathcal{P}^n(\varnothing)$ is most easily defined by simple intensional recursion.

Theorem 31 *(Simple Intensional Recursion). Let $f : \mathscr{U} \longrightarrow \mathscr{U}$ be an intensional function, and let a be a set. Then there is an intensional function $rec(f, a)$ with typing $rec(f, a) : \mathbb{N} \longrightarrow \mathscr{U}$ such that:*

1. $rec(f, a)(0) = a$
2. $rec(f, a)(s(n)) = f(rec(f, a)(n))$.

Furthermore, $rec(f, a)$ is the extensionally unique function satisfying (1) and (2).

Proof. Define $h : \mathscr{U} \times \mathbf{Ord} \longrightarrow \mathscr{U}$ by the lambda expression

$$\lambda(g, n).\; \textbf{if } (n \in \omega \wedge g{:}\mathbf{Fun} \wedge dom(g) = n) \textbf{ then } q(g, n) \textbf{ else } \varnothing \textbf{ fi}$$

where $q(g, n)$ is the term $q(g, n) = \textbf{if } n = 0 \textbf{ then } a \textbf{ else } f(g(p(n))) \textbf{ fi}$. Now, applying the α-recursion theorem to $\alpha = \omega = \mathbb{N}$, we can define $rec(f, a) = rec(h)_\omega : \mathbb{N} \longrightarrow \mathscr{U}$, satisfying

- $rec(f, a)(0) = rec(h)_\omega(0) = a$
- $rec(f, a)(s(n)) = rec(h)_\omega(s(n)) = h(rec(f, a)\restriction_{s(n)}, s(n)) = f(rec(f, a)(n))$,

as desired. Extensional uniqueness then follows from that of $rec(h)_\omega$. □

For example, the function $\lambda n.\; \mathcal{P}^n(\varnothing)$, is just $rec(\mathcal{P}, \varnothing)$. Similarly, even if we had not given a definition of ordinal addition, we could define the function $\lambda n.\; \omega \boxplus n$ recursively in terms of the successor function s as $rec(s, \omega)$. Yet another interesting example of simple intensional recusion is the definition of the set $\mathbb{N}_{\mathscr{Z}}$ of the Zermelo natural numbers, that is, $\mathbb{N}_{\mathscr{Z}} = \{\varnothing, \{\varnothing\}, \{\{\varnothing\}\}, \ldots\}$, which can be defined in one blow by the equality $\mathbb{N}_{\mathscr{Z}} = rec(\{\_\}, \varnothing)[\mathbb{N}]$, where $\{\_\}$ denotes the singleton-forming function $\{\_\} = \lambda X.\; \{X\}$.

An even more interesting example of simple intensional recursion is the construction of the *Zermelo universe* $\mathscr{Z}$, which can be defined in one blow by the equality $\mathscr{Z} = \bigcup rec(\mathcal{P}, \mathbb{N}_{\mathscr{Z}})[\mathbb{N}]$. That is, $\mathscr{Z}$ contains all the sets in $\mathbb{N}_{\mathscr{Z}}$, $\mathcal{P}(\mathbb{N}_{\mathscr{Z}})$, $\mathcal{P}^2(\mathbb{N}_{\mathscr{Z}})$, ..., $\mathcal{P}^n(\mathbb{N}_{\mathscr{Z}})$, and so on. It therefore contains the natural numbers, the real numbers, and in practice all the sets that any mathematician who is not a set theorist might need to use in his/her entire life (see [43] for an excellent in-depth discussion of Zermelo universes). Furthermore, although $\mathscr{Z}$ is just a *set*, and not a proper class, as shown in [43], it satisfies many of the axioms of set theory, including the axioms ($\varnothing$), (*Ext*), (*Sep*), (*Pair*), (*Union*), (*Pow*), (*Inf*), and (*Found*). Therefore, not only does $\mathscr{Z}$ contain all the sets we may ever need, but it is furthermore closed under most of the set-theoretic constructions that we might need in our ordinary mathematical practice.

Exercise 130 *(Transitive Closure). Given any set A, its* transitive closure *$TC(A)$ is the set $TC(A) = \bigcup rec(\bigcup, A)[\mathbb{N}]$, that is, $TC(A) = \bigcup\{A, \bigcup A, \bigcup\bigcup A, \ldots\}$. Prove the following:*

- *$TC(A)$ is a transitive set and $A \subseteq TC(A)$*
- *for any transitive set X, $A \subseteq X \;\Rightarrow\; TC(A) \subseteq X$*
- *X is transitive iff $A = TC(A)$*
- *if $x \in A$, then $TC(x) \subset TC(A)$*
- *$TC(A) = A \cup \bigcup\{TC(x) \mid x \in A\}$.*

18.4.3 Transfinite Recursion

We are now ready to tackle the general case of tranfinitely recursive intensional functions with typing $f : \mathbf{Ord} \longrightarrow \mathscr{U}$. Note that any intensional function f with this typing has also the typing $f : \alpha \longrightarrow \mathscr{U}$ for any ordinal α, and therefore, thanks to the axiom of replacement, defines for each α an *extensional* function, denoted $f\restriction_\alpha$, and called the *restriction* of f to α, namely, the extensional function $f\restriction_\alpha = f : \alpha \twoheadrightarrow f[\alpha]$.

Theorem 32 *(Transfinite Recursion). Let $h : \mathscr{U} \times \mathbf{Ord} \longrightarrow \mathscr{U}$ be an intensional function. Then there exists an intensional function $rec(h)$, with typing $rec(h) : \mathbf{Ord} \longrightarrow \mathscr{U}$ such that*

$$(\forall\beta)\ \beta{:}\mathbf{Ord} \;\Rightarrow\; rec(h)(\beta) = h(rec(h)\restriction_\beta, \beta).$$

Furthermore, $rec(h)$ is extensionally unique*, in the sense that for any other intensional function $f : \mathbf{Ord} \longrightarrow \mathscr{U}$ such that $(\forall\beta)\ \beta{:}\mathbf{Ord} \;\Rightarrow\; f(\beta) = h(f\restriction_\beta, \beta)$, it holds that $(\forall\beta)\ \beta{:}\mathbf{Ord} \;\Rightarrow\; rec(h)(\beta) = f(\beta)$.*

Proof. Extensional uniqueness is easy to prove. Suppose it fails. Then the class of all β such that $rec(h)(\beta) \neq f(\beta)$ is nonempty and has a least element μ. Therefore, by the minimality of μ, $rec(h)|\mu = f\restriction_\mu$. Therefore, $rec(h)(\mu) = h(rec(h)\restriction_\mu, \mu) = h(f\restriction_\mu, \mu) = f(\mu)$, contradicting $rec(h)(\mu) \neq f(\mu)$.

To prove existence, first of all note that, by Lemma 28, an intensional function $f : \mathbf{Ord} \longrightarrow \mathscr{U}$ satisfies $(\forall\beta)\ \beta{:}\mathbf{Ord} \;\Rightarrow\; f(\beta) = h(f\restriction_\beta, \beta)$ iff it satisfies $(\forall\beta)\ \beta{:}\mathbf{Ord} \;\Rightarrow\; f\restriction_\beta = rec(h)_\beta$, where $rec(h)_\beta$ denotes the *extensional* function $rec(h)_\beta : \beta \twoheadrightarrow rec(h)_\beta[\beta]$. This means that we will be done if we can define an intensional function $rec(h)_{\_}$ with typing $rec(h)_{\_} : \mathbf{Ord} \longrightarrow \mathbf{Fun}$ such that for each α, $rec(h)_{\_}(\alpha) = rec(h)_\alpha$, since then we can also define our desired $rec(h) : \mathbf{Ord} \longrightarrow \mathscr{U}$ by the lambda expression

$$rec(h) = \lambda\alpha.\ rec(h)_{s(\alpha)}(\alpha).$$

Indeed, this is the *correct* definition, since for each ordinal α we then have $rec(h)\restriction_\alpha = rec(h)_\alpha$, because for each $\beta \in \alpha$ we have $rec(h)\restriction_\alpha(\beta) = rec(h)_{s(\beta)}(\beta) = rec(h)_\alpha(\beta)$.

Therefore, all boils down to showing the existence of the intensional function $rec(h)_{\_} : \mathbf{Ord} \longrightarrow \mathbf{Fun}$. But note that in the proof of Theorem 30, we proved the existence of the extensional function $rec(h)_\alpha : \alpha \twoheadrightarrow rec(h)_\alpha[\alpha]$ by proving the formula $(\exists! f)\ A(f) \wedge dom(f) = \alpha$, where $rec(h)_\alpha$ was the unique f asserted to exist by that formula, and where the approximation predicate A was defined by the equivalence $A(f) \;\Leftrightarrow\; f{:}\mathbf{Fun} \wedge dom(f) \in s(\alpha) \wedge (\forall\gamma \in dom(f))\ f(\gamma) = h(f\restriction_\gamma, \gamma)$. But $A(f)$ is really a binary predicate in disguise, namely, the predicate $A(f, \alpha)$ defined by the equivalence

$$A(f, \alpha) \;\Leftrightarrow\; f{:}\mathbf{Fun} \wedge dom(f) \in s(\alpha) \wedge (\forall\gamma \in dom(f))\ f(\gamma) = h(f\restriction_\gamma, \gamma).$$

Therefore, since the α in Theorem 30 was *any* ordinal, what we really proved there was the theorem

$$(\forall x)\ x\colon \mathbf{Ord}\ \Rightarrow\ (\exists! y)\ A(y,x).$$

As a consequence, the formula

$$(x\colon \mathbf{Ord}\ \Rightarrow A(x,y))\ \wedge\ (\neg(x\colon \mathbf{Ord})\ \Rightarrow y=\varnothing)$$

provides the correct definition for our desired intensional function $rec(h)\_ : \mathbf{Ord} \longrightarrow \mathbf{Fun}$. □

Exercise 131 *(Ordinal Exponentiation). Transfinite recursion can be used to define* ordinal exponentiation*. As before, to avoid confusion between cardinal and ordinal exponentiation, since they are indeed quite* different *functions, instead of using the standard notation α^β I will use the notation ${}^\beta\alpha$. For each $\alpha \neq 0$ we define an intensional function ${}^{\text{-}}\alpha : \mathbf{Ord} \longrightarrow \mathbf{Ord}$ by means of the recursive definition:*

- ${}^0\alpha = 1$
- ${}^{s(\beta)}\alpha = ({}^\beta\alpha) \otimes \alpha$
- *for β a limit ordinal, ${}^\beta\alpha = \bigcup\{{}^\gamma\alpha \mid \gamma \in \beta\}$.*

Do the following:

1. *Give an explicit definition of the step function h_α such that ${}^{\text{-}}\alpha = rec(h_\alpha)$.*
2. *Use the definition of exponentiation to prove the following properties:*
   - ${}^\beta\alpha \otimes {}^\gamma\alpha = {}^{\beta\boxplus\gamma}\alpha$
   - ${}^\gamma({}^\beta\alpha) = {}^{\beta\otimes\gamma}\alpha$.

An enormously important, yet very simple, use of transfinite recursion is the definition of the *von Neumann universe* $\mathscr{V}$. This is defined using a transfinitely recursive function $\mathscr{V}\_ : \mathbf{Ord} \longrightarrow \mathscr{U}$ with recursive definition:

- $\mathscr{V}_0 = \varnothing$
- $\mathscr{V}_{s(\alpha)} = \mathcal{P}(\mathscr{V}_\alpha)$
- for β a limit ordinal, $\mathscr{V}_\beta = \bigcup\{\mathscr{V}_\alpha \mid \alpha \in \beta\}$.

Then, the von Neumann universe $\mathscr{V}$ is the class defined by the equivalence:

$$x\colon \mathscr{V}\quad \Leftrightarrow\quad (\exists\alpha)\ \alpha\colon \mathbf{Ord}\ \wedge\ x \in \mathscr{V}_\alpha.$$

$\mathscr{V}$ gives us a "constructive" view of sets, where all sets are constructed[4] *ex nihilo* out of the empty set $\varnothing$. As we shall see in §19, the axiom of foundation is precisely the claim that all the sets that exist are constructible sets of this kind, that is, that we have a class equivalence $\mathscr{V} \equiv \mathscr{U}$.

[4] There are at least three, increasingly stronger meanings for "constructive" that must be distinguished: (i) the above, naive notion, in which all is asserted is that sets are *built up* out of $\varnothing$ by the operations $\mathcal{P}$ and $\bigcup$; this first notion is still nonconstructive in at least two ways, because: (a) nothing is said about what counts as a *subset*, and (b) the implicit use of (AC); (ii) the stronger notion of *constructible sets* due to Gödel [20], which is the class $\mathscr{L} \subseteq \mathscr{V} \subseteq \mathscr{U}$ with $\mathscr{L}$ defined by transfinite recursion by the hierarchy: $\mathscr{L}_0 = \emptyset$, $\mathscr{L}_{s(\alpha)} = \mathcal{D}(\mathscr{L}_\alpha)$, and $\mathscr{L}_\beta = \bigcup_{\gamma\in\beta}\mathcal{D}(\mathscr{L}_\gamma)$, for β a limit ordinal, where for each set A, $\mathcal{D}(A) \subseteq \mathcal{P}(A)$ is the set of subsets $B \subseteq A$ *definable* by a set theory formula $\varphi(x, x_1, \ldots, x_n)$, in the sense that there exist elements $a_1, \ldots, a_n \in A$ such that $B = \{a \in A \mid \varphi(a, a_1, \ldots, a_n)\}$; the class $\mathscr{L}$ does not use (AC) in its definition, but satisfies both (AC) and (GCH), which shows that adding these two axioms to ZF does not lead to any inconsistencies; see [15] for a very readable introduction to constructible sets, and [20, 14, 11, 30] for much more thorough treatments; (iii) the even stronger notion of *constructive set* proposed by intuitionistic type theory, in which the law of excluded middle is rejected, and only set-theoretic constructions associated to intutionistic *proofs* are accepted for set formation (see, e.g., [36]).

18.5 Well-Orderings, Choice, and Cardinals

We are going to prove that the axiom of choice (AC) is equivalent to the assertion that every set can be well-ordered, that is, to the axiom

$$(WO) \quad (\forall A)(\exists R \in \mathcal{P}(A \times A)) \;\; (A, R) : \mathbf{WOrd}.$$

The proof of the implication $(AC) \Rightarrow (WO)$ goes back to Zermelo (see [52], 139–141, 183–198). We will then consider cardinals, will show that they form a subclass of the ordinals, and will study some of their properties.

We first prove a lemma.

Lemma 29 *(Hartog's Lemma). For each set A there exists an ordinal α such that there is no injective function $f : \alpha \rightarrowtail A$.*

Proof. If we have an injection $f : \alpha \rightarrowtail A$, then $(f[\alpha], f^2[\supset_\alpha])$ is a well-ordered set. Therefore, we should study the set $WO(\mathcal{P}(A))$ of well ordered sets (B, R) such that $B \subseteq A$. Since $WO(\mathcal{P}(A))$ is a set of pairs, it is a binary relation; in fact, its inverse is precisely the relation

$$WO(\mathcal{P}(A))^{-1} = \{(R, B) \in \bigoplus_{B \in \mathcal{P}(A)} \mathcal{P}(B^2) \mid (B, R) : \mathbf{WOrd}\}.$$

By construction we have $WO(\mathcal{P}(A)) \subseteq \mathbf{WOrd}$. Therefore, by the axiom of replacement we have a set of ordinals $ord[WO(\mathcal{P}(A))]$. I claim that $(ord[WO(\mathcal{P}(A))], \supset_{ord[WO(\mathcal{P}(A))]})$ is itself an ordinal. Since by construction this set is well-ordered, because $ord[WO(\mathcal{P}(A))] \subseteq \mathbf{Ord}$, by Theorem 24 we just need to show that $ord[WO(\mathcal{P}(A))]$ is transitive, so suppose $\alpha \in ord[WO(\mathcal{P}(A))]$ and $\beta \in \alpha$. But $\alpha \in ord[WO(\mathcal{P}(A))]$ exactly means that there is a poset isomorphism $h : (\alpha, \supset_\alpha) \longrightarrow (B, R)$, with $B \subset A$, and since $(\beta, \supset_\beta)$ is an initial segment of $(\alpha, \supset_\alpha)$, by Theorem 21 this means that $(\beta, \supset_\beta)$ is isomorphic to an initial segment of (B, R), and therefore $\beta \in ord[WO(\mathcal{P}(A))]$, as desired. But then the ordinal $ord[WO(\mathcal{P}(A))]$ cannot be mapped injectively to A, since otherwise we would have $ord[WO(\mathcal{P}(A))] \in ord[WO(\mathcal{P}(A))]$, and therefore $ord[WO(\mathcal{P}(A))] \subset ord[WO(\mathcal{P}(A))]$, which forces the contradiction $ord[WO(\mathcal{P}(A))] \neq ord[WO(\mathcal{P}(A))]$. □

Theorem 33 *We have an equivalence $(WO) \Leftrightarrow (AC)$.*

Proof. To see the $(\Rightarrow)$ implication, let A be any set, and let (A, R) be a well-ordering. Then, as already pointed out after Definition 29, the function $\bigwedge_{\_} : \mathcal{P}(A) - \{\varnothing\} \longrightarrow A$ is a choice function.

The $(\Leftarrow)$ implication follows trivially for $A = \varnothing$, so we may assume $A \neq \varnothing$. Let $c : \mathcal{P}(A) - \{\varnothing\} \longrightarrow A$ be a choice function. Then we can define by transfinite recursion an intensional function *fill* with typing $fill : \mathbf{Ord} \longrightarrow A$, which essentially "fills up" the "container" A as if it were a glass of water until it is full. The idea of the function *fill* is very simple: if we have not yet filled A, we pick a new element in the unfilled part of A; otherwise we just throw in the element $c(A)$. On the subclass **Ord**, the function *fill* has the obvious recursive definition[5]

$$fill = \lambda\alpha{:}\mathbf{Ord}.\ \mathbf{if}\ fill[\alpha] \neq A\ \mathbf{then}\ c(A - fill[\alpha])\ \mathbf{else}\ c(A)\ \mathbf{fi}.$$

By Hartog's Lemma 29, the class of ordinals α such that $fill\restriction_\alpha$ is not injective is nonempty; let β be an ordinal in such a class. The only way in which $fill\restriction_\beta$ can fail to be injective is because there are elements $\gamma \in \beta$ such that the condition in the *fill* function fails, that is, such that $fill[\gamma] = A$. Let μ be the smallest such γ. Then, $fill[\mu] = A$, but for each $\delta \in \mu$, $fill[\delta] \neq A$. Therefore, $fill\restriction_\mu$ is injective and obviously surjective, so we have a bijection $fill\restriction_\mu : \mu \longrightarrow A$, and therefore $(A, fill|^2_\mu[\supset_\mu])$ is a well-ordered set, as desired. □

[5]Here and in what follows, I will sometimes define intensional functions just on the class $\mathscr{A}$ we are interested in. It is trivial to explicitly extend such recursive definitions to the whole of $\mathscr{U}$ by adding an additional level of if-then-else nesting that maps each $x : \neg(\mathscr{A})$ to the usual suspect $\varnothing$.

18.5.1 Cardinals

We are now ready to make good on the promise made in §15 of proving the following theorem and elegant construction due to John von Neumann:

Theorem 34 *There is a class* **Card** *of cardinals and an intensional function* $|\_| : \mathscr{U} \longrightarrow \mathbf{Card}$ *such that:*

1. *For A a finite set, $|A|$ is its cardinality in the usual sense of its number of elements.*
2. *For any set A we have $A \cong |A|$.*
3. *For any two sets A and B we have the equivalence $A \cong B \;\Leftrightarrow\; |A| = |B|$.*

Proof. For any set A we can define the set of all well orders on A,

$$\overline{WO}(A) = \{(A, R) \in \{A\} \times \mathcal{P}(A) \mid (A, R) : \mathbf{WOrd}\}.$$

Then we define the intensional function $|\_| : \mathscr{U} \longrightarrow \mathbf{Ord}$ by means of the lambda expression

$$|\_| = \lambda A. \bigwedge ord[\overline{WO}(A)],$$

and define the subclass $\mathbf{Card} \subset \mathbf{Ord}$ by the equivalence

$$x : \mathbf{Card} \quad \Leftrightarrow \quad x : \mathbf{Ord} \wedge x = |x|,$$

so that the function $|\_|$ has also typing $|\_| : \mathscr{U} \longrightarrow \mathbf{Card}$, as desired. Properties (1)–(3) are all trivial consequences of this construction, as is also the property $||A|| = |A|$ for any set A. □

Note that it also follows trivially from the above definition of $|A|$ that we have $A \preceq B$ iff $|A| \subseteq |B|$, and $A \prec B$ iff $|A| \subset |B|$. Yet another trivial consequence of this definition is that, since for any infinite ordinal α we have $|\alpha| = |s(\alpha)|$ and $\alpha \subset s(\alpha)$, any infinite cardinal is a *limit ordinal*. The (CC) property is also a trivial consequence of the definition of $|A|$.

Corollary 9 *(Cardinal Comparability). The axiom* (AC) *implies the axiom* (CC)*:*

(CC) Given any two sets A and B, either $A \preceq B$ or $B \preceq A$.

Proof. By Theorem 22, either $|A| \subseteq |B|$, or $|B| \subseteq |A|$, which gives us either an injective function $A \cong |A| \hookrightarrow |B|$, or an injective function $B \cong |B| \hookrightarrow |A|$, as desired. □

With a little more work one can prove that we have an equivalence $(AC) \Leftrightarrow (CC)$ (see, e.g., [43]).

Recall that we already proved, just from the above properties (1)–(3) about $|\_|$, that the subclass $\mathbf{Card} \subset \mathbf{Ord}$ is proper (see Lemma 15). Note also that **Card** inherits a "well-ordered class" structure $(\mathbf{Card}, \supset)$ from that of $(\mathbf{Ord}, \supset)$. In particular, each nonempty subclass $\mathscr{A} \subseteq \mathbf{Card}$ has a least element $\bigwedge \mathscr{A} : \mathscr{A}$. This allows us to define an intensional function

$$(\_)^+ : \mathbf{Card} \longrightarrow \mathbf{Card}$$

which maps each cardinal κ to the smallest cardinal κ^+ strictly bigger than κ and is defined by the formula

$$(\neg(x : \mathbf{Card}) \Rightarrow y = \varnothing) \wedge (x : \mathbf{Card} \Rightarrow y : \mathbf{Card} \wedge x \prec y \wedge (\forall z)((z : \mathbf{Card} \wedge x \prec z) \Rightarrow y \preceq z)).$$

We call a cardinal κ such that there is a cardinal λ with $\kappa = \lambda^+$ a *successor cardinal*. A cardinal that is not a successor cardinal is called a *limit cardinal*.[6]

Can we construct a "transfinite sequence" that "enumerates" all infinite cardinals? The (positive) answer to this question is precisely the transfinite sequence $\{\aleph_\alpha\}_{\alpha:\mathbf{Ord}}$, which is the intensional function

$$\aleph_\_ : \mathbf{Ord} \longrightarrow \mathbf{InfCard} : \alpha \mapsto \aleph_\alpha,$$

where **InfCard** denotes the class of *infinite cardinals*, and where $\aleph_\_$ is defined by transfinite recursion as follows:

[6] Note that this definition of limit cardinal makes 0 a limit cardinal.

- $\aleph_0 = \omega$
- $\aleph_{s(\alpha)} = |\aleph_\alpha|^+$
- for β a nonzero limit ordinal, $\aleph_\beta = \bigcup\{\aleph_\gamma \mid \gamma \in \beta\}$.

The following theorem shows that the above typing of $\aleph_{\_}$ is correct (this is not entirely obvious, because the third clause defines an ordinal that must be shown to be a cardinal), and that $\aleph_{\_}$ is a stricly monotonic and exhaustive enumeration, and therefore an *isomorphism* of well-ordered classes $(\mathbf{Ord}, \supset) \cong (\mathbf{InfCard}, \supset)$.

Theorem 35 *($\aleph$ Sequence). The intensional function $\aleph_{\_}$ is strictly monotonic. For each $\alpha : \mathbf{Ord}$ we have $\aleph_\alpha : \mathbf{InfCard}$. And for each $\kappa : \mathbf{InfCard}$ there exists a (necessarily unique by strict monotonicity) $\alpha : \mathbf{Ord}$ such that $\kappa = \aleph_\alpha$.*

Proof. The function $\aleph_{\_}$ is clearly monotonic. To prove strict monotonicity, let us assume, towards a contradiction, that there exist $\alpha \subset \beta$ with $\aleph_\alpha = \aleph_\beta$. But then we cannot have $s(\alpha) = \beta$, since this would force $\aleph_\alpha \subset \aleph_{s(\alpha)} = \aleph_\beta$, contradicting $\aleph_\alpha = \aleph_\beta$. So β is a limit ordinal and we have $\aleph_\alpha \subset \aleph_{s(\alpha)} \subseteq \aleph_\beta$, contradicting again $\aleph_\alpha = \aleph_\beta$.

Let us next show that for each $\alpha : \mathbf{Ord}$ we have $\aleph_\alpha : \mathbf{InfCard}$. This is clearly so by construction for $\aleph_0$ and for $\aleph_{s(\alpha)}$. Therefore, if some $\aleph_\beta$ is not a cardinal, β must be a limit ordinal. Let μ be the smallest such limit ordinal. Then, for any $\gamma \in \mu$ we have $\aleph_\gamma = |\aleph_\gamma|$. Since $\aleph_\mu$ is not a cardinal, we have $|\aleph_\mu| \subset \aleph_\mu$. But note that we cannot have $\aleph_\gamma \subset |\aleph_\mu|$ for each $\gamma \in \mu$, since then we would get $\aleph_\mu = \bigcup\{\aleph_\gamma \mid \gamma \in \beta\} \subseteq |\aleph_\mu|$. Therefore, there is a $\gamma \in \mu$ such that $|\aleph_\mu| \subseteq \aleph_\gamma$, and therefore, $|\aleph_\mu| \subseteq \aleph_\gamma \subset \aleph_{s(\gamma)} \subset \aleph_\mu$, which gives us the contradiction $|\aleph_\mu| \leq \aleph_\gamma < \aleph_{s(\gamma)} \leq |\aleph_\mu|$.

Let us show, finally, that for each infinite cardinal κ there exists an ordinal α such that $\kappa = \aleph_\alpha$. Assume that there is a κ outside the range of $\aleph_{\_}$, and let λ be the smallest such cardinal. Obviously, $\lambda \neq 0$, and λ cannot be a successor cardinal, since then it would be $\lambda = (\aleph_\alpha)^+ = \aleph_{s(\alpha)}$ for some α, contradicting λ outside the range of $\aleph_{\_}$. Therefore, λ is a limit cardinal. Now, since the intensional function $\aleph_{\_}$ is strictly monotonic and therefore injective, its defining formula $\varphi(x, y)$ has an inverse $\varphi(x, y)^{-1} = \varphi(y, x)$ which defines a partial intensional function (see Exercise 115) $\aleph_{\_}^{-1} : \aleph_\alpha \mapsto \alpha$, and this partial function is totally defined on the set $\supset_{\mathbf{Card}}[\{\lambda\}]$. Therefore, by the axiom of replacement and Exercise 115, we have a set $\aleph_{\_}^{-1}[\supset_{\mathbf{Card}}[\{\lambda\}]]$, which is exactly the set of all ordinals α such that $\aleph_\alpha \in \lambda$. Let $\beta = \bigcup \aleph_{\_}^{-1}[\supset_{\mathbf{Card}}[\{\lambda\}]]$. Note that β is a limit ordinal, since $\alpha \in \beta$ implies $\aleph_{s(\alpha)} = \aleph_\alpha^+ < \lambda$ by λ limit cardinal, and therefore $s(\alpha) \in \beta$. Also, by construction and by λ outside the range of $\aleph_{\_}$, we have $\aleph_\beta < \lambda$. On the other hand, by the strict monotonicity of $\aleph_{\_}$, we have $\aleph_\alpha < \aleph_\beta$ for each $\alpha \in \beta$, and therefore, $\aleph_\beta \geq \lambda$, a blatant contradiction. □

Note that we can construct another, quite interesting transfinite sequence of cardinals, which starts at ω and jumps from each cardinal κ in the sequence to the cardinal 2^κ, namely, the "beth sequence"

$$\beth_{\_} : \mathbf{Ord} \longrightarrow \mathbf{InfCard} : \alpha \mapsto \beth_\alpha$$

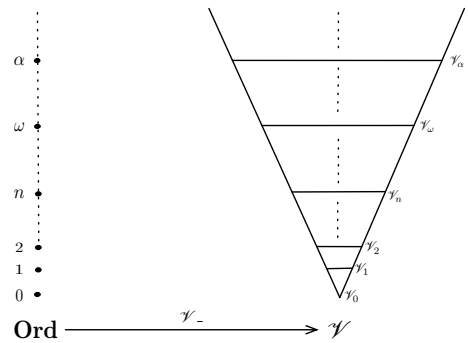
defined recursively by:

- $\beth_0 = \omega$
- $\beth_{s(\alpha)} = 2^{\beth_\alpha}$
- for β a nonzero limit ordinal, $\beth_\beta = \bigcup\{\beth_\gamma \mid \gamma \in \beta\}$.

This sequence is interesting on several counts. First of all, it affords an alternative characterization of (GCH), which is simply the claim $\aleph_{\_} \equiv \beth_{\_}$. Second, it provides a neat characterization of the cardinalities of the different stages $\mathscr{V}_\alpha$ that build up the von Neumann universe $\mathscr{V}$, since it is easy to prove by transfinite induction on α that for each ordinal α we have the equality

$$|\mathscr{V}_{\omega \boxplus \alpha}| = \beth_\alpha.$$

Since the beth sequence is just the transfinite continuation of the *iterated exponential* function $rec(2^{(\cdot)}, 0)$ on natural numbers, where $2^{(\cdot)} = \lambda n \in \mathbb{N}.\ 2^n \in \mathbb{N}$, the size of the sets $\mathscr{V}_\alpha$ grows in an iterated exponential way, $0, 1, 2, 2^2, 2^{(2^2)}, \ldots$ with α, so we can picture the function $\mathscr{V}_{\_}$ in some kind of "iterated logarithmic" scale as filling in the following, increasingly larger triangles.



18.5.2 More Cardinal Arithmetic

Let us pick some low-hanging fruit in the orchard of cardinal arithmetic, thus completing the understanding we gained in §15. Recall that there we proved the following cardinal equalities:

- $\omega + \omega = \omega \cdot \omega = \omega$
- $2^\omega + 2^\omega = 2^\omega \cdot 2^\omega = 2^\omega$
- $\Sigma_{n\in\omega}\omega = \omega$.

This was just the tip of an iceberg, since these equalities hold not just for ω and 2^ω, but for *any* infinite cardinal κ. Futhermore, they are all consequences of a single, general result, namely,

Theorem 36 *For any infinite cardinal κ, we have the equality $\kappa \cdot \kappa = \kappa$.*

Proof. The proof is by transfinite induction. So we assume that for all infinite cardinals $\lambda < \kappa$ we have $\lambda = \lambda \cdot \lambda$ and have to prove that $\kappa = \kappa \cdot \kappa$, which is equivalent to proving $\kappa \geq \kappa \cdot \kappa$. We can define the following well-ordering $\rhd$ on the set $\kappa \times \kappa$:

$$(\gamma, \delta) \rhd (\alpha, \beta) \;\Leftrightarrow\; (\gamma \cup \delta \supset \alpha \cup \beta) \vee (\gamma \cup \delta = \alpha \cup \beta \;\wedge\; (\gamma, \delta)\, Lex(\supset, \supset)\, (\alpha, \beta)).$$

It follows then easily from this definition that for any $(\alpha, \beta) \in \kappa \times \kappa$, we have $|\rhd[\{(\alpha, \beta)\}]| \leq |s(\alpha \cup \beta) \times s(\alpha \cup \beta)| = |s(\alpha \cup \beta)| < \kappa$. Therefore, for each $\nu \in ord(\kappa \times \kappa, \rhd)$, we have $\nu = (\supset[\{\nu\}]) \subset \kappa$, and therefore, $\bigcup ord(\kappa \times \kappa, \rhd) \subseteq \kappa$, which gives us $\kappa \cdot \kappa = |\kappa \times \kappa| = |ord(\kappa \times \kappa, \rhd)| \leq \kappa$, as desired. □

Corollary 10 *Let κ, λ be infinite cardinals. Then:*

1. $\kappa + \lambda = \kappa \cdot \lambda = \kappa \cup \lambda$
2. $\Sigma_{n\in\omega}\kappa = \kappa$.

Proof. (1) follows from $(\kappa \cup \lambda) = (\kappa \cup \lambda) \cdot (\kappa \cup \lambda)$, plus the chain of inequalities $\kappa \cup \lambda \leq \kappa + \lambda \leq \kappa \cdot \lambda \leq (\kappa \cup \lambda) \cdot (\kappa \cup \lambda)$. (2) follows from (1), plus the trivial observation (Exercise 90–(1)), that $\bigoplus_{n\in\omega} \kappa = \kappa \times \omega$. □

Exercise 132 *(Generalizes Exercise 107). Prove that for any infinite cardinal κ and any cardinal λ such that $\kappa \geq \lambda$, we always have $\kappa + \lambda = \kappa$.*

Exercise 133 *Prove that for any infinite cardinal κ we have the equality $\Sigma_{n\in\omega}\kappa^n = \kappa$.*

Recall that in §15 we also proved that for each $n \in \omega$, with $2 \leq n$ we have the equalities $2^\omega = n^\omega = \omega^\omega$. This was also the tip of another huge iceberg, namely, the much more general result:

Theorem 37 *For cardinals κ, λ with $2 \leq \kappa \leq \lambda$, and λ infinite, we have the equalities*

$$2^\lambda = \kappa^\lambda = \lambda^\lambda = (2^\lambda)^\lambda.$$

Proof. By monotonicity of exponentiation we have the inequalities $2^\lambda \leq \kappa^\lambda \leq \lambda^\lambda \leq (2^\lambda)^\lambda$. Therefore, it suffices to prove the equality $2^\lambda = (2^\lambda)^\lambda$. But, since by Theorem 36 we have $\lambda = \lambda \cdot \lambda$, and therefore a bijection $[\lambda \to 2] \cong [\lambda \times \lambda \to 2]$, and by currying (see Exercise 46) we have the bijection $[\lambda \times \lambda \to 2] \cong [\lambda \to [\lambda \to 2]]$, we get the cardinal equality $2^\lambda = (2^\lambda)^\lambda$, as desired. □

Exercise 134 *Prove that any infinite cardinal κ satisfies the equality $(\kappa^+)^\kappa = 2^\kappa$. (Hint: Use Theorem 37).*

18.5.3 Regular, Singular, and Inaccessible Cardinals

Since any infinite cardinal κ is a limit ordinal, we have the equality $\kappa = \bigcup_{\alpha\in\kappa}\alpha$; and, of course, if $\alpha \in \kappa$, the inequality $|\alpha| < \kappa$. Therefore, κ can be obtained as the union of a κ-indexed family of sets of cardinality strictly smaller than κ. This raises the question: could κ be obtained as a union of a λ-indexed family of ordinals strictly smaller than κ, where λ is a cardinal such that $\lambda \leq \kappa$? That, is, can we find cardinals λ with $\lambda \leq \kappa$ and functions $f : \lambda \longrightarrow \kappa$ such that $\kappa = \bigcup_{\alpha\in\lambda} f(\alpha)$? Let us call a cardinal $\lambda \leq \kappa$ *cofinal* for κ iff such an f exists. Obviously, κ is cofinal for itself with $f = id_\kappa$. Therefore, the set of cardinals λ that are cofinal for κ is nonempty and has a least element. We call this least element the *cofinality* of κ and denote it $cof(\kappa)$. So an interesting question, then, is: for what infinite cardinals κ do we have $cof(\kappa) < \kappa$? Let us call an infinite cardinal κ *regular* iff $cof(\kappa) = \kappa$, and *singular* iff $cof(\kappa) < \kappa$.

Note that, since any finite union of finite sets is a finite set, we have $cof(\aleph_0) = \aleph_0$, and therefore $\aleph_0$ is regular. Similarly, since by Corollary 4 we know that countable unions of countable sets are countable, $\aleph_1$ is also regular. What about infinite successor cardinals in general?

Lemma 30 *For any infinite cardinal κ, its successor cardinal κ^+ is regular.*

Proof. Consider the folowing sequence of cardinal inequalities:

$$\kappa^+ = \bigcup_{\alpha\in cof(\kappa^+)} f(\alpha) \leq \Sigma_{\alpha\in cof(\kappa^+)}|f(\alpha)| \leq \Sigma_{\alpha\in cof(\kappa^+)}\kappa = cof(\kappa^+)\cdot\kappa = cof(\kappa^+)\cup\kappa = cof(\kappa^+) \leq \kappa^+$$

which forces $cof(\kappa^+) = \kappa^+$, as desired. □

Therefore, all cardinals in the countable sequence $\aleph_0, \aleph_1, \aleph_2, \ldots, \aleph_n, \aleph_{n+1}, \ldots$, that is, all infinite cardinals that any mathematician who is not a set theorist is likely to encounter in his/her entire life, are regular. Obviously, all singular cardinals are limit cardinals. But are all infinite limit cardinals singular? Certainly not, since $\aleph_0$ is regular. But are there any *uncountable* limit cardinals that are regular? Nobody knows. In fact, we *know that we don't know*, since it can be proved that *ZFC cannot prove* the existence of an uncountable, regular limit cardinal (see [30], Corollary VI 4.13). An uncountable, regular limit cardinal is called a *weakly inaccessible cardinal*. The fact that we cannot prove in *ZFC* that they exist does not rule them out. Actually, many set theorists believe that assuming their existence is intuitively plausible and should not lead to any contradiction (see, e.g., the discussions in [15, 11]); and that one could likewise assume the existence of a *strongly inaccessible cardinal*, which is a weakly inaccessible cardinal λ such that

$$\kappa < \lambda \;\Rightarrow\; 2^\kappa < \lambda.$$

A strongly inaccessible cardinal λ is attractive from a conceptual standpoint because assuming its existence one can prove that the *set* $\mathscr{V}_\lambda$ is closed under all the set-theoretic constructions, including the powerset construction, and satisfies all the axioms of *ZFC*; so $\mathscr{V}_\lambda$ is an attractive *universe* within which we can carry out all mathematical practice, including all set-theoretic practice.

Chapter 19

Well-Founded Sets and The Axiom of Foundation

Well-founded sets can be understood from the top down or from the bottom up. The axiom of foundation states that all sets are well-founded.

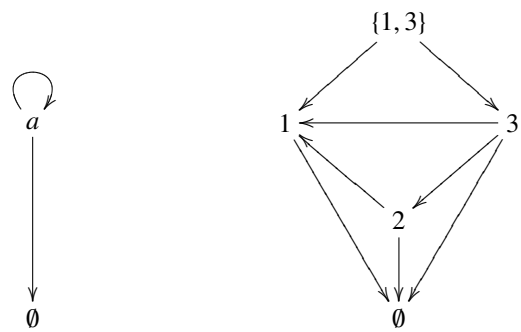
19.1 Well-Founded Sets from the Top Down

There are various versions of the tale about an old oriental cosmogony according to which the entire world rests on the back of a tiger, which rests on the back of an elephant, which, in turn, rest on the back of a turtle. In the tale, somebody asks a famous guru the question: "And on top of what is the turtle resting?" The guru hesitates for a moment and then answers:

> *"It is turtles all the way down."*

This tale has a quite direct bearing on our mental picture of sets. Recall the metaphor, developed in Footnote 1 of §3.3, of sets as boxes, whose elements are other boxes they contain inside. Imagine a game in which, after opening a box, one chooses one of the boxes inside, opens it, chooses another box inside this second box, opens it, and keeps going until ... when? If it is "turtles all the way down," this game may go on forever, without ever reaching a bottom. There is nothing in the axioms of set theory we have seen so far preventing the possibility of having a set a that is an element of itself, so that, for example, we may have $a = \{a, \emptyset\}$. Although the order of the elements inside a set does not matter, let us go on with our metaphor and assume that the box a is rectangular, so that when we open it, the box a is on the left, and the box $\emptyset$ is on the right. If the player chooses to open the righthand box at any moment in the game, the game will be over, since when opening the box $\emptyset$ there will be literally nothing in it. However, if the player's strategy is to always choose the leftmost box, the game will never end.

This box metaphor is of course much more than a metaphor. It can be made both mathematically precise and *graphical* by observing that $(\mathscr{U}, \ni)$ is a "mega-relation," and therefore a "megagraph." Therefore, we can picture each set x as a *node* in the megagraph $(\mathscr{U}, \ni)$, and then represent the relation $x \ni y$ as an *edge* $x \to y$ in this megagraph. As a consequence, the elements of a set x are precisely the nodes y that have an edge $x \to y$. For example, the sets $a = \{a, \emptyset\}$, and $\{1, 3\}$ can be depicted by the graphs:



where the graph on the right depicts not just the membership relation for the elements $1, 3$ in the set $\{1, 3\}$ (the edges $\{1, 3\} \to 1$ and $\{1, 3\} \to 3$), but all the memberships in its transitive closure $TC(\{1, 3\})$ (note that $a = TC(a)$). There is nothing intrinsically wrong or "pathological" with sets like a. On the contrary, it is possible to adopt an "Anti-Foundation Axiom" (*AFA*) which celebrates this kind of recursive circularity in sets, and exploits it in many applications, including quite interesting computer science ones (see [1, 5], and, for shorter introductions, [43, 15]). Sets with such circularities, or, more generally, sets where we may never "bottom out," are called *ill-founded sets*. Instead, sets where the box-opening game always stops no matter what inner boxes we choose are called *well-founded sets*. As we shall see shortly, well-founded sets form a subclass $\mathbf{WF} \subseteq \mathscr{U}$ of the entire universe of sets $\mathscr{U}$. Although ill-founded sets can indeed be quite useful, it seems fair to say that:

1. In normal mathematical practice the sets we encounter are well-founded; and for the usual mathematical structures (groups, posets, topological spaces, etc.), even if the original structure is based on an ill-founded set, we can find an *isomorphic* structure based on a well-founded one.

2. It is possible to model non-well-founded sets using well-founded ones; specifically, as a subclass $\mathscr{U}_{AFA} \subset \mathbf{WF}$, which, together with a specially-defined membership relation ϵ_{AFA}, gives us a "mega-graph" $(\mathscr{U}_{AFA}, \epsilon_{AFA})$, which, interpreting set membership $\in$ as the relation ϵ_{AFA}, satisfies all the axioms of non-well-founded sets (see [43], Appendix B, for a very accessible and careful account of the non-well-founded universe $(\mathscr{U}_{AFA}, \epsilon_{AFA})$ within $\mathbf{WF}$).

3. There are some definite advantages of working in the class $\mathbf{WF}$, including, as we shall see: (i) that $\mathbf{WF} = \mathscr{V}$, so our "constructive" picture of sets as built up out of the empty set $\varnothing$ by union and powerset operations is confirmed by this choice; and (ii) that $\mathbf{WF}$ comes with a quite useful *induction principle*, called $\ni$-*induction*, which generalizes transfinite induction on ordinals and can be used to prove set properties.

So, how can we characterize the class $\mathbf{WF}$? It is of course the class of sets x that always "bottom out," that is, sets x for which there is no infinite descending chain of memberships

$$x \ni x_1 \ni x_2 \ni \ldots x_n \ni x_{n+1} \ni \ldots$$

Can this be stated in a more intrinsic way as a property of the set x itself? Our first guess might be to use the property that $(x, \ni_x)$ is a *well-founded relation*. This is clearly necessary, because if $(x, \ni_x)$ is not a well-founded relation, it will have an infinite descending chain $x_1 \ni x_2 \ni \ldots x_n \ni x_{n+1} \ni \ldots$, yielding also an infinite descending chain $x \ni x_1 \ni x_2 \ni \ldots x_n \ni x_{n+1} \ni \ldots$. However, it is not sufficient. For example, for $a = \{a, \varnothing\}$, the singleton set $\{\{a\}\}$ has no edge in the empty relation $\ni_{\{\{a\}\}} = \{(x, y) \in \{\{a\}\}^2 \mid x \ni y\}$, and is therefore a well-founded relation, yet, we have the infinite descending chain $\{\{a\}\} \ni \{a\} \ni a \ni a \ni \ldots \ni a \ni a \ldots$ The point is that we cannot look at the set x in isolation: we have to "dig deeper" inside its elements, that is, we should consider not just x but its *transitive closure* $TC(x)$.

Lemma 31 *The following are equivalent for any set x:*

1. *There is no countable sequence of sets $\{x_n\}_{n \in \mathbb{N}}$ such that $x = x_0$, and for each $n \in \mathbb{N}$ we have $x_n \ni x_{s(n)}$.*

2. *$(TC(x), \ni_{TC(x)})$ is a well-founded relation.*

Proof. To see (2) $\Rightarrow$ (1), first of all observe that (1) is just the set theory formula $(\nexists f)\ f : \mathbf{Fun} \wedge p_1[f] = \mathbb{N} \wedge f(0) = x \wedge (\forall n \in \mathbb{N})\ f(n) \ni f(s(n))$. But, by Exercises 125–(2) and 130, if there is a function f such that $f(0) = x \wedge (\forall n \in \mathbb{N})\ f(n) \ni f(s(n))$, then f is such that $(\forall n \in \mathbb{N})\ f(n) \in TC(x)$, which makes $(TC(x), \ni_{TC(x)})$ a non-well-founded relation. To see (1) $\Rightarrow$ (2), we reason by contradiction and observe that $(TC(x), \ni_{TC(x)})$ a non-well-founded relation implies the existence of a countable descending chain $x_0 \ni x_1 \ni x_2 \ni \ldots x_n \ni x_{n+1} \ni \ldots$, plus the fact that either $x_0 \in x$, or there is a $k > 0$ such that $x_0 \in \bigcup^k x$, which means that we have a finite chain $x \ni y_0 \ni y_1 \ni \ldots y_k$, with $y_k = x_0$, and therefore, also a countable descending chain $x \ni y_0 \ni y_1 \ni \ldots y_k \ni x_1 \ni x_2 \ni \ldots x_n \ni x_{n+1} \ni \ldots$, as desired. □

Since (1) $\Leftrightarrow$ (2), and both (1) and (2) can be expressed as set theory formulas, we can use either of them to define our desired class **WF** of well-founded sets. Using (2), we can give the following definition:

$$x : \mathbf{WF} \quad \Leftrightarrow \quad (TC(x), \ni_{TC(x)}) : \mathbf{WFRel}.$$

The axiom of foundation (*Found*), is just the statement $(\forall x)\ x : \mathbf{WF}$, and will be discussed in detail in §19.3. However, until §19.3 we will *not assume* the (*Found*) axiom, that is, the equivalence $\mathbf{WF} \equiv \mathscr{U}$, in our reasoning. Instead, we will reason about the properties of the class **WF** without necessarily assuming that it is the class of all sets. This is useful to investigate other characterizations of the (*Found*) axiom, such as the equivalence $\mathscr{V} \equiv \mathscr{U}$. To be clear about this, I will use the abbreviation ZFC^- to refer to all the axioms of *ZFC except for* (*Found*), and will preface all results prior to §19.3 with (ZFC^-), since we are not assuming (*Found*) in their proof. In fact, some results follow from substantially weaker axioms than ZFC^-.

Exercise 135 (ZFC^-) *Prove the following for any set x:*

1. $x : \mathbf{WF} \Leftrightarrow TC(x) : \mathbf{WF}$.
2. *If* $x : \mathbf{WF}$ *and* $x \ni y$*, then* $y : \mathbf{WF}$.
3. *If* $x : \mathbf{WF}$*, then* $s(x) : \mathbf{WF}$.

19.1.1 $\ni$-Induction

Note the interesting fact that, since each ordinal α is transitive (so $\alpha = TC(\alpha)$), and well-ordered (and therefore well-founded) by $\ni_\alpha$, all ordinals are well-founded sets. Therefore, we have a class inclusion $\mathbf{Ord} \subseteq \mathbf{WF}$. Furthermore, since by Lemma 31, if $x : \mathbf{WF}$, then there is no countable sequence of sets $\{x_n\}_{n\in\mathbb{N}}$ such that $x = x_0$, and for each $n \in \mathbb{N}$ we have $x_n \ni x_{s(n)}$, we should think of $(\mathbf{WF}, \ni)$ as a *well-founded intensional relation*, which both generalizes and extends the "well-ordered megachain" $(\mathbf{Ord}, \ni)$. This suggests the following, straightforward generalizations of Theorems 28 and 29, which can both be proved in (ZFC^-).

Theorem 38 (ZFC^-) *Each nonempty* subclass $\mathscr{A} \subseteq \mathbf{WF}$ *has a* $\ni$*-minimal element, that is, a set x such that* $x : \mathscr{A}$ *and for any y with* $x \ni y$*, then* $y : \neg\mathscr{A}$.

Proof. Suppose that $\mathscr{A} \subseteq \mathbf{WF}$ is nonempty, so there is a set A with $A : \mathscr{A}$. Consider now the set $B = \{x \in TC(s(A)) \mid x : \mathscr{A}\}$. Since $A \in B$, B is nonempty, and $B \subseteq TC(s(A))$. If we show that $(TC(s(A)), \ni_{TC(s(A))})$ is a well-founded relation we will be done, since then there is a $b \in B$ (and therefore $b : \mathscr{A}$), such that $\ni_{TC(s(A))}[\{b\}] \cap B = \emptyset$, and then if we assume, towards a contradiction, that there is a $y : \mathscr{A}$ with $b \ni y$, then, by $TC(s(A))$ transitive, we have $y \in TC(s(A))$, and therefore $y \in B$, which is impossible, since $\ni_{TC(s(A))}[\{b\}] \cap B = \emptyset$. So, all we have left to do is to show that $(TC(s(A)), \ni_{TC(s(A))})$ is a well-founded relation. Note that $A : \mathscr{A} \subseteq \mathbf{WF}$, and that we will be done if we show $s(A) : \mathbf{WF}$. But this follows from Exercise 135–(3). □

Theorem 39 *($\ni$-Induction).* (ZFC^-) *Let* $\mathscr{A} \subseteq \mathbf{WF}$ *be a subclass such that for each* $A : \mathbf{WF}$*, the implication* $((\forall x \in A)\ x : \mathscr{A}) \Rightarrow A : \mathscr{A}$ *holds. Then* $\mathscr{A} \equiv \mathbf{WF}$.

Proof. Suppose not, so that $\mathbf{WF} - \mathscr{A}$ is nonempty. Let A be a $\ni$-minimal element in $\mathbf{WF} - \mathscr{A}$; then, by the minimality of A and Exercise 135–(2), we get $(\forall x \in A)\ x : \mathscr{A}$, and therefore $A : \mathscr{A}$, contradicting $A : (\mathbf{WF} - \mathscr{A})$. □

Is there also a $\ni$*-recursion theorem* generalizing and extending to the well-founded class $(\mathbf{WF}, \ni)$ the transfinite recursion theorem (Theorem 32) for the well-ordered megachain $(\mathbf{Ord}, \ni)$? The answer is yes! In

fact, an even more general R-recursion theorem can be proved for any "set-like" well-founded class $(\mathscr{A}, R)$, that is, a class $\mathscr{A}$ together with a well-founded intensional relation R that is "set-like" in the precise sense that there is an intensional function $R[\{\_\}] : \mathscr{A} \longrightarrow \mathscr{U}$ such that $(\forall x, y)\ (x, y : \mathscr{A} \ \wedge\ x\,R\,y) \ \Leftrightarrow\ y \in R[\{x\}]$, that is, the class of all y in $\mathscr{A}$ "R-below" a given $x : \mathscr{A}$ is actually a set; note that this is indeed the case for the $\ni$ relation on **WF**. The proof of this R-recursion theorem is a generalization of that for Theorem 32 (see, e.g., [30], Thm. III.5.6, or [29], Thm. 5.4).

19.2 Well-Founded Sets from the Bottom Up

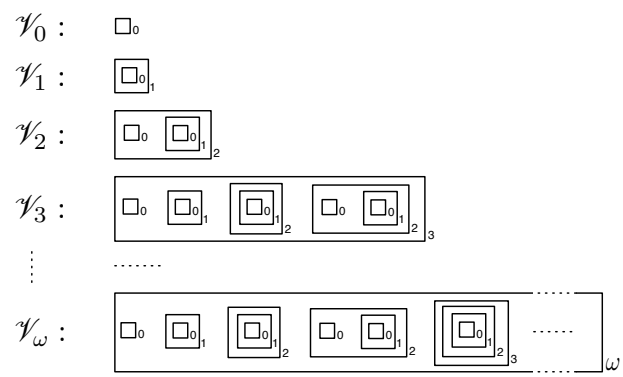
Our view of well-founded sets so far has been "from the top down," that is, we begin with a set as a box and go down inside its inner boxes until we bottom out in the empty set. It is interesting to think about well-founded sets in an alternative "bottom up," "constructive" fashion, where we begin with the set at the bottom, namely $\varnothing$, and build up all the other sets by powerset and union operations. The key question, of course, is: are these top-down and bottom-up pictures *equivalent*? That is, do they describe the *same* class of sets? The answer is yes! so that we indeed have the class equivalence $\mathbf{WF} \equiv \mathscr{V}$.

To gain an intuitive feel for why this is so, we may go back to our "sets as boxes" metaphor (in essence our graph-theoretic view of sets), and think of the construction of the von Neumann hierarchy of sets $\mathscr{V}$ as the work of an incredibly skilled and infinitely fast Hungarian carpenter called Johnny. Johnny wears a digital watch, but his life is eternal, so that at each moment, when he looks at his watch, he does not see just a natural number n, corresponding to the number of seconds elapsed since he started building $\mathscr{V}$, but an *ordinal* α. For example, if he sees $\omega + 7$ in his watch display, this means that the time is seven seconds after the *first* eternity ω. At each moment, Johnny builds a new box out of the boxes he has already built before. He can do this extremely fast, so that in just one second he is able to build the new box needed at that moment. Also, when he builds the box for time α, he *writes the time* at which he built it on the outside of the box. Furthermore, Johnny has *perfect memory*, so that he remembers all the boxes he has already built; this means that if at some later time he needs to build a *copy*[1] of a box he had already built at a previous time β, he will mark that copy with the earliest time β at which the original box now being copied was built. Let us call $\mathscr{V}_\alpha$ the box that Johnny builds at time α. Such a box can be easily described:

- $\mathscr{V}_0$: here Johnny just builds a single empty box and writes 0 on it.
- $\mathscr{V}_{\alpha+1}$: here Johnny has already built the box $\mathscr{V}_\alpha$ and now builds a new box $\mathscr{V}_{\alpha+1}$, which he marks with its time $\alpha + 1$ and inside which he places, for each subcollection B of boxes among those found inside $\mathscr{V}_\alpha$ (set-theoretically, for each subset $B \subseteq \mathscr{V}_\alpha$), a box containing (an exact copy of) each box in B. If the box thus obtained for B is one he had not built before, he writes on it the label α that was written on the bigger box $\mathscr{V}_\alpha$; otherwise, he marks it with the earliest time β at which the original being copied was built.
- $\mathscr{V}_\lambda$: if Johnny just arrived at some eternal time λ (a limit ordinal), he builds a new box $\mathscr{V}_\lambda$, marked with time λ, inside which he empties the contents of all the boxes that he had built in all previous moments before reaching that eternal time (but avoiding duplicate copies).

Note that it follows immediately from this construction that all the boxes Johnny ever builds are well founded, because any player in the box-opening game who begins with a box built by Johnny will always bottom out regardless of which inner boxes he chooses to open. This is so because the player can keep a list of the labels α marked on the boxes he/she has opened so far, and as he/she keeps opening boxes, gets a descending chain of ordinals $\alpha_0 \ni \alpha_1 \ni \alpha_2 \ni \ldots \alpha_k$, which must bottom out at some finite k with $\alpha_k = 0$, precisely because the ordinals are well-ordered. Therefore, we obviously have $\mathscr{V} \subseteq \mathbf{WF}$. The rest of this section is just crossing the t's and dotting the i's in the above, intuitive picture of the containment $\mathscr{V} \subseteq \mathbf{WF}$, and getting the converse containment $\mathbf{WF} \subseteq \mathscr{V}$. The stages $\mathscr{V}_0, \mathscr{V}_1, \mathscr{V}_2, \mathscr{V}_3, \ldots \mathscr{V}_\omega$, in Johnny's box-building effort are depicted below.

[1] Recall that in the physical metaphor of boxes, the same box cannot be simultaneously inside two different boxes, so copies of it may be needed. Instead, in set theory, by the (*Ext*) axiom, the same box is always *shared*, and can therefore be inside many other boxes; in particular, the empty box $\varnothing$ is inside all other boxes. Graph-theoretically, the difference between boxes and sets is just the difference between a tree and its maximally-shared directed acyclic graph (DAG) representation.



Recall again the recursive definition of the von Neumann hierarchy $\mathscr{V}$: (i) $\mathscr{V}_0 = \varnothing$, (ii) $\mathscr{V}_{s(\alpha)} = \mathcal{P}(\mathscr{V}_\alpha)$, and (iii) for λ a limit ordinal, $\mathscr{V}_\lambda = \bigcup_{\gamma\in\lambda} \mathscr{V}_\gamma$. It follows immediately from (i) and (iii) that for any set x such that $x : \mathscr{V}$, the smallest ordinal α such that $x \in \mathscr{V}_\alpha$ must be a successor ordinal. The following notion of rank of a set $x : \mathscr{V}$ exactly corresponds to the label β that Johnny marks on the box x in our sets-as-boxes metaphor.

Definition 33 *Given a set $x : \mathscr{V}$, its* rank *is the smallest ordinal β such that $x \in \mathscr{V}_{s(\beta)}$. We then write $rank(x) = \beta$.*

Therefore, if $rank(x) = \beta$, we have $x \subseteq \mathscr{V}_\beta$ and $x \notin \mathscr{V}_\beta$, which was exactly the case when Johnny used the label β to mark the box x, since x was either $\mathscr{V}_\beta$ or a new, smaller box not built earlier (i.e., $x \notin \mathscr{V}_\beta$). By definition we also have $x \in \mathscr{V}_{s(\beta)}$, and, more generally, $x \in \mathscr{V}_\gamma$ for any γ such that $\gamma \ni \beta$, as shown by the following lemma.

Lemma 32 (ZFC^-) *For any ordinal α, $\mathscr{V}_\alpha$ is a transitive set; furthermore, for any ordinal γ such that $\gamma \in \alpha$ we have a strict inclusion $\mathscr{V}_\gamma \subset \mathscr{V}_\alpha$.*

Proof. The proof of the conjunction of both statements is by transfinite induction on α. For $\alpha = 0$ the result is obvious. For $\alpha = s(\beta)$ we have $\mathscr{V}_\beta$ transitive by the induction hypothesis, and therefore $\mathscr{V}_{s(\beta)} = \mathcal{P}(\mathscr{V}_\beta)$ is transitive by Exercise 125–(5), and $\mathscr{V}_\beta \subset \mathcal{P}(\mathscr{V}_\beta)$ by Exercise 125–(4), which trivially implies $\mathscr{V}_\gamma \subset \mathscr{V}_{s(\beta)}$ for any $\gamma \in s(\beta)$ by the induction hypothesis. For λ a limit ordinal we have $\mathscr{V}_\lambda = \bigcup_{\gamma\in\lambda} \mathscr{V}_\gamma$, which by the induction hypothesis and Exercise 125–(6) makes $\mathscr{V}_\lambda$ transitive. Also, we trivially have $\mathscr{V}_\gamma \subset \mathscr{V}_\lambda$ for any $\gamma \in \lambda$, since we have $\mathscr{V}_\gamma \subset \mathscr{V}_{s(\gamma)} \subseteq \mathscr{V}_\lambda$ by the induction hypothesis and the definition of $\mathscr{V}_\lambda$. □

Another interesting feature of Johnny's box-building method is that at any time α, the box $\mathscr{V}_\alpha$ contains inside only and exactly those boxes labeled with a β strictly smaller than α.

Lemma 33 (ZFC^-) *For any ordinal α, we have*

$$(\forall x)\ x \in \mathscr{V}_\alpha \quad \Leftrightarrow \quad x : \mathscr{V} \ \wedge\ rank(x) < \alpha.$$

Proof. Suppose $x : \mathscr{V}$ and α is an ordinal; then the proof is just the following chain of obvious equivalences:

$$x \in \mathscr{V}_\alpha \quad \Leftrightarrow \quad (\exists \beta \in \alpha)\ x \in \mathscr{V}_{s(\beta)} \subseteq \mathscr{V}_\alpha \quad \Leftrightarrow \quad rank(x) \in \alpha. \ \square$$

The following lemma collects a couple of useful and intuitively obvious facts about ranks.

Lemma 34 (ZFC^-) *Let $x : \mathscr{V}$, then:*

1. $(\forall y \in x)\ rank(y) < rank(x)$
2. $rank(x) = \bigcup\{s(rank(y)) \mid y \in x\}$.

Proof. To see (1), let $\alpha = rank(x)$. Then $x \in \mathscr{V}_{s(\alpha)} = \mathcal{P}(\mathscr{V}_\alpha)$. Therefore, if $y \in x$ we have $y \in \mathscr{V}_\alpha$, and therefore, $rank(y) < \alpha = rank(x)$ by Lemma 33.

To see (2), first note that (1) implies $(\forall y \in x)\ s(rank(y)) \subseteq rank(x)$. Therefore, $\alpha = \bigcup\{s(rank(y)) \mid y \in x\} \subseteq rank(x)$. Furthermore, by Lemma 33 we have $x \subseteq \mathscr{V}_\alpha$, and therefore $x \in \mathscr{V}_{s(\alpha)}$, thus $rank(x) \subseteq \alpha$, which gives us (2). □.

We are now ready for the main theorem of this section.

Theorem 40 (ZFC^-) *The classes of well-founded sets and of the von Neumann universe are* equivalent*, that is, we have* $\mathbf{WF} \equiv \mathscr{V}$.

Proof. To see $\mathscr{V} \subseteq \mathbf{WF}$, just note that if $x : \mathscr{V}$, then we cannot have an infinite descending chain $x \ni x_1 \ni x_2 \ni \ldots \ni x_n \ni x_{n+1} \ni \ldots$, since by Lemma 34–(1) this would give us an infinite descending chain of ordinals $rank(x) \ni rank(x_1) \ni rank(x_2) \ni \ldots \ni rank(x_n) \ni rank(x_{n+1}) \ni \ldots$, which is impossible. This is just crossing the t's and dotting the i's in our box-game argument showing the well-foundedness of Johnny's boxes.

To see $\mathbf{WF} \subseteq \mathscr{V}$, we can reason by $\ni$-induction, that is, for $x : \mathbf{WF}$, we assume that for each $y \in x$ we have $y : \mathscr{V}$ and then show that $x : \mathscr{V}$. Let $\alpha = \bigcup\{s(rank(y)) \mid y \in x\}$; then by Lemma 33 we have $x \subseteq \mathscr{V}_\alpha$, and therefore $x \in \mathscr{V}_{s(\alpha)}$, as desired. □

Of course, since we already know that $\mathbf{Ord} \subset \mathbf{WF}$, by Theorem 40 we also have the inclusion $\mathbf{Ord} \subset \mathscr{V}$. But we can be more precise.

Lemma 35 (ZFC^-) *For each ordinal α we have:*

1. $rank(\alpha) = \alpha$
2. $\alpha = \{x \in \mathscr{V}_\alpha \mid x : \mathbf{Ord}\}$.

Proof. We can prove (1) by transfinite induction on α; so we assume $rank(\beta) = \beta$ for $\beta \in \alpha$ and must show $rank(\alpha) = \alpha$. But $rank(\beta) = \beta$ forces $\beta \in \mathscr{V}_{s(\beta)} \subseteq \mathscr{V}_\alpha$, and therefore $\alpha \subseteq \mathscr{V}_\alpha$, which gives us $\alpha \in \mathscr{V}_{s(\alpha)}$, thus, $rank(\alpha) \subseteq \alpha$. On the other hand, by Lemma 34, we have $rank(\alpha) = \bigcup\{s(rank(\beta)) \mid \beta \in \alpha\} = \bigcup\{s(\beta) \mid \beta \in \alpha\} \subseteq \alpha$, which gives us $rank(\alpha) = \alpha$, as desired.

To see (2), note that by the definition of ordinals and Lemma 33, we have $\alpha = \{\beta \mid \beta \in \alpha\} = \{\beta \mid rank(\beta) < \alpha\} = \{x \in \mathscr{V}_\alpha \mid x : \mathbf{Ord}\}$, as desired. □

19.3 The Axiom of Foundation

The *Axiom of Foundation* (sometimes also called the *Axiom of Regularity*) is the statement

All sets are well-founded,

which can be specified as the formula

$$(\mathit{Found}) \qquad (\forall x)\ x{:}\mathbf{WF}$$

or, equivalently, as the class equivalence $\mathbf{WF} \equiv \mathscr{U}$. It is however worthwhile to study other, equivalent formulations of the (*Found*) axiom. Here are some.

Theorem 41 (ZFC^-) *The following formulas are equivalent formulations of the* (*Found*) *axiom:*

1. $(\forall x)\ x{:}\mathbf{WF}$
2. $(\forall x)\ (TC(x), \ni_{TC(x)}){:}\mathbf{WFRel}$
3. $(\forall x)\ (\nexists f)\ (f{:}\mathbf{Fun} \wedge p_1[f] = \mathbb{N} \wedge f(0) = x \wedge (\forall n \in \mathbb{N})\ f(n) \ni f(s(n)))$
4. $(\forall x)\ (x, \ni_x) : \mathbf{WFRel}$
5. $(\forall x)\ ((\exists u)\ u \in x) \Rightarrow (\exists y)(y \in x \wedge \neg(\exists z)(z \in x \wedge z \in y))$

6. $(\forall x)\ x \neq \emptyset \ \Rightarrow\ (\exists y)(y \in x \ \wedge\ x \cap y = \emptyset)$

7. $(\forall x)\ x : \mathscr{V}$.

Proof. The equivalence (1) $\Leftrightarrow$ (2) is just defining away the class **WF**; and the equivalence (1) $\Leftrightarrow$ (7) is Theorem 40. The equivalence (2) $\Leftrightarrow$ (3) is Lemma 31. The equivalence (5) $\Leftrightarrow$ (6) is trivial. The implication (4) $\Rightarrow$ (2) is also trivial; and (2) $\Rightarrow$ (4) follows directly from the observation that $x \subseteq TC(x)$, so that if a well-founded $(TC(x), \ni_{TC(x)})$ is restricted to $(x, \ni_x)$, we get also a well-founded relation. So it is enough to show (4) $\Leftrightarrow$ (5). But note that (5) can be equivalently expressed as the formula $(\forall x)\ (x \neq \emptyset) \ \Rightarrow\ ((\exists y \in x)\ \ni_x[\{y\}] = \emptyset)$. To prove (4) $\Rightarrow$ (5), we assume (4) and reason by cases. The implication holds trivially in the case $x = \emptyset$; if we instead have $x \neq \emptyset$, then since $(x, \ni_x)$ is well-founded and $x \neq \emptyset$, we obviously get (5). Conversely, if (5) holds, then for any set A we have $(A, \ni_A)$ well-founded, because for any $x \subseteq A$ with $x \neq \emptyset$, there exists a $y \in x$ with $\ni_x[\{y\}] = \emptyset$ by (5). □

Note that (5) is expressed in the basic language of set theory with no extra symbols at all; so it is often preferred as the standard formulations of (*Found*). However, (3) seems the most intuitive formulation of (1)–(6), and can also be easily defined away into a formula involving only the = and $\in$ predicates. Note also that the formula $(\forall x)\ x \notin x$ follows trivially from (*Found*), so that all sets are noncircular.

Could adding the axiom of foundation lead to any contradiction? That is, if the axioms ZFC^- are consistent (do not lead to any contradiction) are the axioms ZFC consistent? The answer to this second questions is yes! In this regard, Theorem 40 is quite important, because it gives us a "constructive" view of all well-founded sets as built up "ex nihilo" out of the empty set. This makes it relatively easy to show that **WF**, that is, the von Neumann universe $\mathscr{V}$, which we can define in ZFC^- without any recourse to (*Found*), is *closed* under all set-theoretic constructions (that is, we never leave the universe $\mathscr{V}$ when building other sets out of those already in $\mathscr{V}$), so that $\mathscr{V}$ itself, and not just $\mathscr{U}$, satisfies all the axioms of ZFC^-; but since it also satisfies (*Found*), we then get that if the axioms ZFC^- are consistent, then the axioms ZFC are consistent. Technically, showing that $\mathscr{V}$ satisfies all the axioms of ZFC^- requires *relativizing* those axioms to the class $\mathscr{V}$, that is, we have to restrict the universal and existential quantification in the axioms of ZFC^- to range only over sets in $\mathscr{V}$.[2] This can be easily done for any formula φ in the language of set theory to obtain a relativized sentence $\varphi^{\mathscr{V}}$, since whenever we have a formula $(\forall x)\ \varphi$, we can recursively replace it by $(\forall x)\ (x : \mathscr{V}) \Rightarrow \varphi^{\mathscr{V}}$, and whenever we have a formula $(\exists x)\ \varphi$, we recursively replace it by $(\exists x)\ (x : \mathscr{V}) \wedge \varphi^{\mathscr{V}}$, and leave all the Boolean connectives and basic predicates = and $\in$ untouched. For example, the relativization of the $(\emptyset)$ axiom to $\mathscr{V}$ is the formula $(\emptyset)^{\mathscr{V}} = (\exists x)(x : \mathscr{V}) \wedge ((\forall y)\ (y : \mathscr{V}) \Rightarrow y \notin x)$. I refer the reader to [30], §IV.4, and [11], §II.5, for detailed proofs of the fact that the universe $\mathscr{V}$ satisfies all the axioms of ZFC^- and that, therefore, if the axioms ZFC^- are consistent, then the axioms ZFC are consistent. However, the essential idea needed to get these results is just the *closure* of $\mathscr{V}$ under the usual set-theoretic operations.

Exercise 136 *Prove that, given $x, y : \mathscr{V}$:*

1. $\bigcup x, \mathcal{P}(x), \{x\} : \mathscr{V}$

2. $x \cup y, x \cap y, x \times y, \{x, y\}, (x, y), [x \to y] : \mathscr{V}$.

Prove also that $\mathbb{N}, \mathbb{Z}, \mathbb{R} : \mathscr{V}$ (for $\mathbb{R}$ you can just use the binary representation of the reals from §15.3).

Exercise 137 *(ZFC^-) Let (A, R) be a binary relation (e.g., a poset, lattice, well-founded relation, etc.). Show that there is a binary relation (A', R') such that $(A', R') : \mathscr{V}$ and we have a relation isomorphism $h : (A, R) \longrightarrow (A', R')$. This is just the tip of a huge iceberg: any mathematical structure we may think of: a group, a topological space, a Hilbert space, and so on, is likewise isomorphic (for the appropriate notion of structure-preserving isomorphism) to one in $\mathscr{V}$. Therefore, the (Found) axiom, although not needed for mathematical practice, imposes no limits on such practice.*

Exercise 138 *Prove that the model $(\mathscr{V}_\omega, \{\in_{\mathscr{V}_\omega}\})$ defined in §5.4 satisfies the (Found) axiom.*

Let me finish this chapter with an elegant and very useful technical application of the (*Found*) axiom known as "Scott's trick," because it was proposed by the well-known logician and computer scientist Dana Scott. As explained in §9.6, given a set A and an equivalence relation $\equiv$ on it, we can form the *quotient set* $A/\equiv$ and get a quotient map $q_\equiv : A \twoheadrightarrow A/\equiv$: $a \mapsto [a]_\equiv$. Can this be done for classes? That is,

[2] This is just like what happens when proving that a subset $A \subseteq G$ of some structure G such as, say, a group, is also a group: we have to show that A is closed under all the group operations and prove the group axioms for A, thus restricting their quantification to elements of A.

given a class $\mathscr{A}$ and an *intensional* equivalence relation $\approx$ on it (that is, an intensional relation such that whenever $x, y, z : \mathscr{A}$, we then have $x \approx x$, $x \approx y \Rightarrow y \approx x$, and $(x \approx y \wedge y \approx z) \Rightarrow x \approx z$), can we *define* a class denoted $\mathscr{A}/\approx$ and a surjective intensional function $q_{\approx} : \mathscr{A} \twoheadrightarrow \mathscr{A}/\approx$, such that for each $x : \mathscr{A}$, $q_{\approx}(x)$ is its corresponding $\approx$-equivalence class? In general we cannot, because $q_{\approx}(x)$ must be a *set*, but the $\approx$-equivalence class of x may easily be a *proper class*. For example, the bijectivity or equinumerosity relation $x \cong y$ between sets is obviously an intensional equivalence relation on both $\mathscr{U}$ and $\mathscr{V}$, but for any set $A \neq \emptyset$, the sets B such that $A \cong B$ form a proper class. Dana Scott's insight was to point out that, since by (*Found*) we have $\mathscr{U} \equiv \mathscr{V}$, we can get exactly what we want by observing that each $\approx$-equivalence class will have a representative of *minimum rank*, that is, we have an intensional function $\mu_{\approx} : \mathscr{A} \longrightarrow \mathbf{Ord} : x \mapsto \bigwedge\{\alpha \in s(rank(x)) \mid (\exists y)\ y : \mathscr{A} \wedge y \approx x \wedge \alpha = rank(y)\}$. We can then define the $\approx$-equivalence class of $x : \mathscr{A}$ as the *set* $[x]_{\approx} = \{y \in \mathscr{V}_{s(\mu_{\approx}(x))} \mid y : \mathscr{A} \wedge y \approx x\}$. This then defines the class $\mathscr{A}/\approx$ by the equivalence

$$x : \mathscr{A}/\approx \ \Leftrightarrow\ (\exists y)\ (y : \mathscr{A}) \wedge x = [y]_{\approx}$$

and then the intensional function $\lambda x : \mathscr{A}.\ [x]_{\approx} : \mathscr{A}/\approx$ gives us our desired surjection $q_{\approx} : \mathscr{A} \twoheadrightarrow \mathscr{A}/\approx$.

Exercise 139 *Prove in detail that for the intensional equivalence relation $x \cong y$, the mapping card $: [x]_{\cong} \mapsto |x|$ is an intensional function and defines a* bijection *of classes card $: (\mathscr{U}/\cong) \longrightarrow$ **Card**. That is, you should exhibit a formula $\varphi(x, y)$ corresponding to the above assignment notation $[x]_{\cong} \mapsto |x|$, show that it is an intensional function with typing card $: (\mathscr{U}/\cong) \longrightarrow$ **Card**, and prove that it is bijective for that typing. Note that the bijection card is conceptually quite useful, since it expresses the essential identity between two, somewhat different notions of cardinal: one, more abstract and due to Cantor, as the abstract concept associated to an equivalence class of sets having the same size, and another, more concrete and due to von Neumann, as a canonical representative of such an equivalence class.*

Exercise 140 *(Factorization Theorem for Intensional Functions (see Corollary 1)). For any intensional function f with typing $f : \mathscr{A} \longrightarrow \mathscr{B}$, Prove that:*

1. *The intensional relation $\approx_f$ defined by the logical equivalence $x \approx_f x' \Leftrightarrow f(x) = f(x')$ is an intensional equivalence relation on $\mathscr{A}$.*
2. *The class $f[\mathscr{A}]$, defined by the logical equivalence $x : f[\mathscr{A}] \Leftrightarrow (\exists z)\ z : \mathscr{A} \wedge f(z) = x$, has a subclass inclusion $f[\mathscr{A}] \subseteq \mathscr{B}$.*
3. *There is an intensional function $\widehat{f}$, which has typing $\widehat{f} : \mathscr{A}/\approx_f \longrightarrow f[\mathscr{A}]$, and is* bijective *for that typing and such that the diagram*

$$\begin{tikzcd}
\mathscr{A} \arrow[r, "f"] \arrow[d, two heads, "q_{\approx_f}"'] & \mathscr{B} \\
\mathscr{A}/\approx_f \arrow[r, "\widehat{f}"] & f[\mathscr{A}] \arrow[u, hook, "id"']
\end{tikzcd}$$

with $id = \lambda x.\ x$, commutes, in the sense that we can prove $(\forall x)\ x : \mathscr{A} \Rightarrow f(x) = \widehat{f}(q_{\approx_f}(x))$.

*Apply (3) to the surjective intensional function $|\_| : \mathscr{U} \twoheadrightarrow$ **Card** to obtain a trivial proof of Exercise 139.*

Exercise 141 *Use Exercise 140 to prove that the surjective intensional function ord $:$ **WOrd** $\twoheadrightarrow$ **Ord** is such that:*

1. *the intensional equivalence relation $\approx_{ord}$ is precisely the intensional relation $\cong$ of poset isomorphism in* **WOrd**
2. *the bijective intensional function $\widehat{ord}$ $:$ **WOrd**$/\cong \longrightarrow$ **Ord** is actually an isomorphism of well-ordered classes $\widehat{ord} : (\mathbf{WOrd}/\cong, >) \longrightarrow (\mathbf{Ord}, \supset)$, where $>$ is the relation induced on **WOrd**$/\cong$ by the order relation $>$ on* **WOrd** *defined in §18.2, so that $q_{\cong} : (\mathbf{WOrd}, >) \twoheadrightarrow (\mathbf{WOrd}/\cong, >)$ is strictly monotonic.*

Note that we can view (2) as a vindication of Cantor's notion of ordinal as the abstract concept associated to an isomorphism class $[(A, >)]_{\cong}$ of well-ordered sets, since up to isomorphism of well-ordered classes, von Neumann's construction of $(\mathbf{Ord}, \supset)$ completely agrees with Cantor's definition of the relation $\alpha > \beta$ between ordinals in [9] §14.

Bibliography

[1] P. Aczel. *Non-Well-Founded Sets*. CSLI Publications, 1988.

[2] J. Avigad and S. Feferman. Gödel's functional ("Dialectica") interpretation. In S. Buss, editor, *Handdbook of Proof Theory*, pages 337–405. Elsevier, 1998.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[4] J. Barwise. An introduction to first-order logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 5–46. North-Holland, 1977.

[5] J. Barwise and L. Moss. *Vicious Circles*. CSLI Publications, 1996.

[6] J. Bergstra and J. Tucker. Characterization of computable data types by means of a finite equational specification method. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, Seventh Colloquium*, pages 76–90. Springer-Verlag, 1980. LNCS, Volume 81.

[7] G. Birkhoff. *Lattice Theory*. American Mathematical Society, 1948.

[8] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.

[9] G. Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers*. Dover, 1955. Originally published in *Math. Annalen* XLVI, 481–512, 1895, and XLIX, 207–246, 1897.

[10] M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude – A High-Performance Logical Framework*. Springer LNCS Vol. 4350, 2007.

[11] P. Cohen. *Set Theory and the Continuum Hypothesis*. W.A. Benjamin, 1966.

[12] M. Davis. *Computability and Unsolvability*. Dover, 1982.

[13] M. Davis, editor. *The Undecidable — Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Dover, 2004.

[14] K. Devlin. Constructibility. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 453–489. North-Holland, 1977.

[15] K. Devlin. *The Joy of Sets – Fundamentals of Contemporary Set Theory*. Springer Verlag, 1993.

[16] F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 21(1-2):59–88, 2008.

[17] F. Durán, S. Lucas, and J. Meseguer. MTT: The Maude Termination Tool (system description). In *IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 313–319. Springer, 2008.

[18] F. Durán and J. Meseguer. A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. in Proc. *WRLA'10*, Springer LNCS 6381, 2010.

[19] S. Feferman. *In the Light of Logic*. Oxford U.P., 1998.

[20] K. Gödel. *The Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory*. Princeton U.P., 1940. Annals of Mathematical Studies, Vol. 3.

[21] K. Gödel. What is Cantor's continuum problem? *American Mathematical Monthly*, 54:515–525, 1947. Revised and expanded in P. Benaceraff and H. Putnam, *Philosophy of Mathematics — Selected Readings*, Cambridge UP, 1983, 470–485.

[22] J. Goguen. Principles of parameterized programming. In T. Biggerstaff and A. Perlis, editors, *Software Reusability, Volume I: Concepts and Models*, pages 159–225. Addison-Wesley, 1989.

[23] N. Goldenfeld and C. Woese. Biology's next revolution. `http://www.hhmi.org/news/lindquist2.html`, 2007.

[24] P. Halmos. *Naive Set Theory*. Van Nostrand, 1960.

[25] J. Hendrix, J. Meseguer, and H. Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006*, pages 151–155, 2006.

[26] J. Hendrix, H. Ohsaki, and J. Meseguer. Sufficient completeness checking with propositional tree automata. Technical report, CS Dept. Univ. Illinois at Urbana-Champaign, 2005. `http://www.ideals.illinois.edu/handle/2142/11096`.

[27] H. Hermes. *Enumerability, Decidability, Computability*. Springer Verlag, 1969.

[28] T. Jech. About the axiom of choice. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 345–370. North-Holland, 1977.

[29] P. Johnstone. *Notes on Logic and Set Theory*. Cambridge University Press, 1987.

[30] K. Kunen. *Set Theory – An Introduction to Independence Proofs*. Elsevier, 1980.

[31] F. W. Lawvere. An elementary theory of the category of sets. *Proceedings, National Academy of Sciences*, 52:1506–1511, 1964.

[32] F. W. Lawvere. The category of categories as a foundation for mathematics. In S. Eilenberg, D. Harrison, S. MacLane, and H. Röhrl, editors, *Proc. Conf. Categorical Algebra, La Jolla 1965*, pages 1–20. Springer-Verlag, 1966.

[33] F. W. Lawvere. Continuously variable sets: Algebraic Geometry = Geometric Logic. In *Proc. Logic Colloquium (Bristol 1973)*, pages 135–153. North-Holland, 1975.

[34] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic*. Springer Verlag, 1992.

[35] P. Manolios and D. Vroon. Ordinal arithmetic: Algorithms and mechanization. *J. Autom. Reasoning*, 34(4):387–423, 2005.

[36] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[37] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Commun. ACM*, 3(4):184–195, 1960. Available at `http://www-formal.stanford.edu/jmc/`.

[38] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirshberg, editors, *Computer Programming and Formal Systems*. North-Holland, 1963. Available at `http://www-formal.stanford.edu/jmc/`.

[39] J. McCarthy, P. Abrahams, D. Edwards, T. Hart, and M. Levin. *LISP 1.5 Programmer's Manual*. MIT Press, 1965.

[40] J. Meseguer. Order completion monads. *Algebra Universalis*, 16:63–82, 1983.

[41] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, pages 18–61. Springer LNCS 1376, 1998.

[42] J. Meseguer and J. Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge University Press, 1985.

[43] Y. Moschovakis. *Notes on Set Theory*. Springer Verlag, 2006.

[44] P. D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B, Chapter 11*. North-Holland, 1990.

[45] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, 1967.

[46] B. Russell. *Introduction to Mathematical Philosophy*. George Allen & Unwin, 1920.

[47] D. Scott. Outline of a mathematical theory of computation. In *Proceedings, Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176. Princeton University, 1970. Also appeared as Technical Monograph PRG 2, Oxford University, Programming Research Group.

[48] D. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In *Microwave Research Institute Symposia Series, Vol. 21: Proc. Symp. on Computers and Automata*. Polytechnical Institute of Brooklyn, 1971.

[49] J. R. Shoenfield. *Degrees of Unsolvability*. North-Holland, 1971.

[50] A. Stepanov and P. McJones. *Elements of Programming*. Addison wesley, 2009.

[51] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

[52] J. van Heijenoort, editor. *From Frege to Gödel — A Source Book in Mathematical Logic, 1879–1931*. Harvard Univ. Press, 1967.

[53] P. Winkler. *Mathematical Mind-Benders*. A. K. Peters Ltd., 2007.

[54] W. H. Woodin. The continuum hypothesis, parts I–II. *Notices of the AMS*, 48:567–576, and 681–690, 2001.

[55] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.