Set Theory and Algebra in Computer Science

A Gentle Introduction to Mathematical Modeling

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

August 28, 2013

Contents

Part I

Basic Set Theory

Chapter 1

Introduction to Part I

"... 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 [47], 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 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 [47], 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 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, and I-indexed sets 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.

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 will be 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 several exercises.

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., [29, 30] and [32] 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 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, 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*). All of them, except for (*Rep*) and (*Found*), 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 §**??**).

[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 an horizontal bar yields a new theorem, which we can derive as a *logical consequence* of theorems 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 [3].

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 Part I we will follow the standard mathematical practice of giving rigorous but informal proofs. However, in Part II proofs in equational logic (a sublogic of first-order logic) will be fully formalized, and the *soundness and completeness* of equational logic (the fact that it can prove *only* true theorems, and *all* true theorems) will be proved in detail.

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 (∅) 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 ∅. 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 ∅ 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 ∅ 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 ⇒ in the formula, instead of the equivalence ⇔, 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 ⇐ for free, so that it needs not be explicitly stated in the axiom. Note that, as already mentioned, extensionality makes sure that our ∅ 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 [47], 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 [47], 104–112), 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 [47], 150–182). A simpler solution was given the same year by Zermelo (see [47], 199–215), and was subsequently formalized and refined by Skolem (see [47], 290–301), and Fraenkel (see [47], 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:

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

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 [47], 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 is the basis of 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 is of course 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 models 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 to Hausdorff, see [47] 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 29, and in §5.5 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 informally 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:

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

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 an unordered 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 $x = \emptyset$, 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 an unordered 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* $\bigcup \varnothing = \varnothing$*, and that for any nonempty 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 $A \cap B = \varnothing$. For an arbitrary set of sets[2] X, the corresponding, most useful notion of disjointness is not just requiring $\bigcap X = \varnothing$, 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 $x \neq y \Rightarrow x \cap y = \varnothing$. 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)* $X = Y = \varnothing$*; or (ii)* $X \neq \varnothing \wedge \varnothing \notin X$*. That is, a partition* X *of* $Y = \bigcup X$ *is either the empty collection of sets when* $Y = \varnothing$*, 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 $\bigcap U = \varnothing$, 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, nonempty 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 $\varnothing$, $\cup$, $\cap$, 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*

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

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:*

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

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 $0 = \emptyset$, and $n+1 = s(n) = n \cup \{n\}$. If we unpack this definition, the von Neumann natural number sequence looks as follows:

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

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 $\emptyset$, the singleton sets $\{0\}$, $\{1\}$, $\{2\}$, and the unordered pairs $\{0, 1\}$, $\{0, 2\}$, and $\{1, 2\}$. That is,

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

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 informally 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 $\mathcal{U} = \emptyset$, 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: $\bigcap \emptyset = X$. 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 $\emptyset \subseteq \{U\}$, we should always have $\bigcap \emptyset \supseteq \bigcap \{U\} = U$. Since we know that the biggest set in $\mathcal{P}(X)$ is X itself, it is then entirely natural to define $\bigcap \emptyset = X$, 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:*

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

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 $U \cap \varnothing = \varnothing$ 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 $\varnothing$ 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 $\mathcal{P}(\varnothing)$, $\mathcal{P}(\mathcal{P}(\varnothing))$, and $\mathcal{P}(\mathcal{P}(\mathcal{P}(\varnothing)))$.*

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 $(A \times \{0\}) \cap (B \times \{1\}) = \varnothing$. 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 $A = \varnothing$, 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 8 in §8.

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 , the powersets $\mathcal{P}(7)$ and $\mathcal{P}(\mathcal{P}(7))$, 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, …, n, …, 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 informally stated in 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.*
- *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* $\mathcal{P}(S)$*:*

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

[4] In what follows, all numbers will always be understood to be represented as sets in the von Neumann representation.

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 §6.3.

Chapter 5

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.

5.1 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*[2] (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." Sometimes an inverse relation R^{-1} is suggestively denoted by the mirror image of the symbols for R. For example, $>^{-1}$ is denoted $<$, and $\geq^{-1}$ is denoted $\leq$. 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, a function 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*

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

[2] This does not necessarily imply that for each $b \in B$ there is an $a \in A$ with $(a, b) \in R$.

from A to *B*. Can we define $[A \to B]$ precisely? Yes, of course:

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

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 $sin(\pi)$, $factorial(7)$, $2 + 2$ (which in the above prefix notation would be rendered $+(2, 2)$), and so on.

Exercise 20 *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.*

5.2 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.[3] 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[4] $y^2 = x$, 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

[3] 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 $(a_1, (a_2, \ldots, a_n))$.

[4] 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.

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 also use the above formula notation to specify functions; but this is not always a good idea. 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 $x \subseteq y \;\Leftrightarrow\; (\forall z)(z \in x \Rightarrow z \in y)$. 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 21 *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.*

5.3 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 22 *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. Give 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[5] *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: (i) $R \subseteq R^\bullet$, and (ii) 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 23 *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 $\_{\restriction_{A'}} \colon [A \to B] \longrightarrow [A' \to B]$. Show, using the inclusion $\mathbb{N} \subset \mathbb{Z}$ of the natural numbers into the set $\mathbb{Z}$ of integers, as well as Exercise 22-(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 $\_{\restriction_{A'}} \colon \mathcal{P}(A \times B) \longrightarrow \mathcal{P}(A' \times B)$.

5.4 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 2 *(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 $(F; G); H = F; (G; H)$.*

[5] 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 7.4. 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 should always make clear which of these two different senses of "total" or "partial" is meant.

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 22-(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 23. 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 can also be written $j^A_{A'} : A' \rightarrowtail A$, but the notation $j^A_{A'} : A' \hookrightarrow A$ is clearly more informative, since it indicates that $j^A_{A'}$ is not only injective, but also an inclusion.

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 22, 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, for any B such that $f[A] \subseteq B$ 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 necessarily* 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 24 *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 24 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."

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 25 *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 §10.2.

Exercise 26 *(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,[6] 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 $1 = \{\emptyset\}$. 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 35 shows that we always have a bijection $A \cong [1{\to}A]$.

Exercise 27 *Prove the following for any set A:*

[6] 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."

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 $id_\varnothing$.

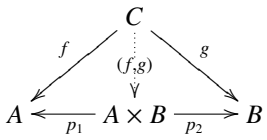
Given any two sets A and B, we can associate to their cartesian product two functions, $p_1 : A \times B \longrightarrow A : (a, b) \mapsto a$ and $p_2 : A \times B \longrightarrow B : (a, b) \mapsto b$, which are called the *projection* functions from the cartesian product $A \times B$ into their first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

Exercise 28 *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 29 *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* $h = (f, g)$.

We can compactly express properties (1)–(3) in Exercise 29 in a precise graphical notation by means of the following *commutative diagram:*



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).

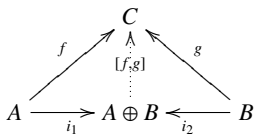
Given any two sets A and B, we can associate to their disjoint union $A \oplus B$ two injective functions, $i_1 : A \longrightarrow A \oplus B : a \mapsto (a, 0)$ and $i_2 : B \longrightarrow A \oplus B : b \mapsto (b, 1)$, which are called the *injection* functions into the disjoint union $A \oplus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

Exercise 30 *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:



Note the striking similarity between Exercises 29 and 30, 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 6 to Execise 26, 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.

5.5 Abstract Products and Disjoint Unions

Exercises 30 and 29 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 29 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 29 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 2 *(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 29 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 2 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 28), this does of course also satisfy the abstract requirements in Definition 2. 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 35, 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 2, 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 2:

Definition 3 *(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 30 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 3 by taking $[f, g] = f \cup g$. This representation is so common that a special notation is used: we write $A \uplus B$ to denote $A \cup B$ when $A \cap B = \emptyset$. 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 31 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 31 *(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 32 *($\oplus$ 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 component, 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 33 *(***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.)*

5.6 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 34 *($[\_{\to}\_]$ 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 (and "covariant" in the second 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 35 *Prove the following for any set A:*

1. *The first projection map $p_A : A \times 1 \longrightarrow A : (a, \varnothing) \mapsto a$ 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 $[1{\to}A] \longrightarrow A : f \mapsto f(\varnothing)$ 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 $\widehat{a}(\varnothing) = a$, and, conversely, any function $f \in [1{\to}A]$ is precisely of the form $f = \widehat{a}$ for a unique $a \in A$.*

Exercise 36 *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 35-(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 20). 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 37 *(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

$$uncurry : [A \to [B \to C]] \longrightarrow [A \times B \to C] : h \mapsto (h \times id_B); \_(\_)_{[B\to C]}$$

where $h \times id_B : A \times B \longrightarrow [B \to C] \times B$ is the unique function associated to h and id_B in Exercise 32.

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, $uncurry(h)(x, y) = h(x)(y)$. 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.

Let me finish this Chapter 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 38 *(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 6

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 Richard Dedekind and Giuseppe Peano, and reformulated in category theory terms by F. William Lawvere, which are intimately related to simple recursion.

6.1 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, due to Richard Dedekind [11, §IX], shows.

Theorem 3 *(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:

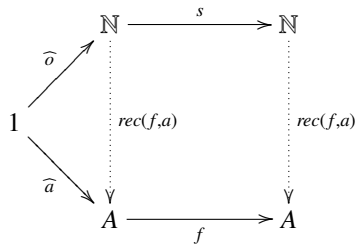
1. $(0, a) \in R$
2. $(\forall n \in \mathbb{N})\ ((n, a') \in R \ \Rightarrow (s(n), f(a')) \in R)$.

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 reformulated as the statement that the set $\mathbb{N}$ of von Neumann natural numbers satisfies the following more general axiom due to F. William Lawvere [29]. Recall from Exercise 35 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, which I call the *Dedekind-Lawvere*[1] axiom.

> *(Dedekind-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:*



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

Obviously, Theorem 3 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 *Dedekind-Lawvere* axiom. But note that the *Dedekind-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 *Dedekind-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 Dedekind-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 §5.5, where, similarly, we viewed a product or a disjoint union as an abstract data type and became liberated from any representation bias.

[1] It is generally called the "Peano-Lawvere" axiom. But this is clearly unfair to Dedekind, whose work Peano knew and cited, and who, to the best of my knowledge, was the first to state and prove Theorem 3. Furthermore, Dedekind's definition of the natural numbers is an *abstract* one, not depending on any particular representation, but "up to isomorphism," exactly as captured by the Dedekind-Lawvere axiom.

In Exercise 31 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 39 *(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* Dedekind-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 40 *(Dedekind-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 Dedekind-Lawvere axiom also satisfies the following induction property: $(\forall T \subseteq \mathbb{N})\ ((0 \in T \ \wedge\ ((\forall x \in T)\ s(x) \in T)) \ \Rightarrow T = \mathbb{N})$.*

6.2 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 $k : \mathbb{N} \xrightarrow{\cong} \mathbb{N} \times 1 : n \mapsto (n, \varnothing)$ 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 4 *(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 "uncurrying" $rec([B \to f], g)$ (see Exercise 37). That is, we define $prec(f, g) = uncurry(rec([B \to f], g))$. 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 $+ = uncurry(rec([\mathbb{N}\rightarrow s], id_{\mathbb{N}}))$.

Exercise 41 *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 29) 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., [26], §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 $[f\rightarrow g]$, 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 3 *If currying and uncurrying of functions (plus composition, tupling, identities, projections, $\_(\_)$, column, and $[f\rightarrow g]$ for previously defined f, g), are 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 $\widehat{A} = curry(A) : \mathbb{N} \longrightarrow [\mathbb{N}\rightarrow\mathbb{N}]$. 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 $Iter : [\mathbb{N}\rightarrow\mathbb{N}] \longrightarrow [\mathbb{N}\rightarrow\mathbb{N}] : f \mapsto \lambda n.\, f^{s(n)}(1)$. 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 $Iter = \widehat{Iter_0}; [\mathbb{N}\rightarrow p_2]$, where $\widehat{Iter_0}$ is the currying of $Iter_0$, and where $Iter_0$ is itself defined by primitive recursion as $Iter_0 = prec((p_1, \_(\_)), (id_{[\mathbb{N}\rightarrow\mathbb{N}]}, \_(1})))$, where $(p_1, \_(\_)) : [\mathbb{N}\rightarrow\mathbb{N}] \times \mathbb{N} \longrightarrow [\mathbb{N}\rightarrow\mathbb{N}] \times \mathbb{N} : (f, n) \mapsto (f, f(n))$, and $(id_{[\mathbb{N}\rightarrow\mathbb{N}]}, \_(1)) : [\mathbb{N}\rightarrow\mathbb{N}] \longrightarrow [\mathbb{N}\rightarrow\mathbb{N}] \times \mathbb{N} : f \mapsto (f, f(1))$. □

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 $[\mathbb{N}\rightarrow\mathbb{N}]$, $[[\mathbb{N}\rightarrow\mathbb{N}]^2\rightarrow[\mathbb{N}^3\rightarrow\mathbb{N}]]$, 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 [1], 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. §7.5 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 §11.

6.3 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 *Dedekind-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 *Dedekind-Lawvere* axiom (part of this proof is already contained in Exercise 40). This, in turn, will imply it for the von Neumann representation thanks to Theorem 3.

In fact, we can prove more than that. As implicit already in [29], and shown in Prop. 17 of [38] as a consequence of a more general result that applies not only to the natural numbers but to any so-called initial algebra,[2] we have in fact an *equivalence*:

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

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 *Dedekind-Lawvere* axiom.

Exercise 42 *Give two set theory formulas expressing precisely the Peano axioms (as a single formula) and the Dedekind-Lawvere axiom (again, as a single formula).*

[2] The *Dedekind-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. Initial algebras will be studied in detail in Part II.

Chapter 7

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 22-(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 $R \in \mathcal{P}(A \times A)$.

7.1 Directed and Undirected Graphs

The first obvious thing to observe is that a *directed graph* G with set of nodes N and a binary relation (N, G) 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 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 43 *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* $n \geq 1$, $G^n \cap id_N = \emptyset$.

Exercise 44 *For each* $n \in \mathbb{N} - \{0\}$, *the (standard)* cycle *function is defined for* $n = 1$ *as* $cy_1 = id_1$, *and for all other* n *by* $cy_n = \{(0, 1), (1, 2), \ldots, (n-2, n-1)\}$. *Likewise, if* A *is a finite, nonempty set with* n *elements, we call* $f : A \longrightarrow A$ *a* cycle *iff there is a bijection* $g : A \longrightarrow n$ *such that* $f = g; cy_n; g^{-1}$. *Notice that in the light of Exercise 43–(5), when* f *is viewed as a graph, calling* f *a cycle is an exact and perfect description of* f.

Prove the following:

1. *(Cycle Decomposition of any Permutation). Given any finite, nonempty set* A *and a bijection* $f : A \longrightarrow A$, *there is a subset* $\{A_1, \ldots, A_k\} \subseteq (\mathcal{P}(A) - \{\emptyset\})$ *such that: (i)* $A = \bigcup\{A_1, \ldots, A_k\}$; *(ii) for each* i, j *such that* $1 \leq i < j \leq k$, $A_i \cap A_j = \emptyset$; *and (iii) there are cycles* $f_i : A_i \longrightarrow A_i$, $1 \leq i \leq k$, *such that* $f = f_1 \cup \ldots \cup f_k$. (**Hint**: *For an easy proof, use Exercise 43–(5));*

2. *Show that there is an* $n \in \mathbb{N} - \{0\}$ *such that* $f^n = id_A$. *Give an arithmetic characterization of the smallest such* n *in terms of the numbers of elements in the sets* $A_1, \ldots, A_k$. *Note that this shows that, if* A *is finite, the inverse* f^{-1} *of any bijection* $f : A \longrightarrow A$ *is a power of* f.

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 should 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

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

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

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

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 45 *Give a necessary and sufficient condition* φ *on* A *and* B, *so that the surjective function* $und : A \times B \ \longrightarrow\!\!\!\!\!\rightarrow A \otimes B$ *is bijective iff* φ *holds. Note that if* φ *holds, then the surjective function* $und[\_] : \mathcal{P}(A \times B) \ \longrightarrow\!\!\!\!\!\rightarrow \mathcal{P}(A \otimes B)$ *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 46 *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.*

7.2 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 the transition system viewpoint, 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 4 *A binary relation $R \in \mathcal{P}(A \times A)$ is called* reflexive *iff $id_A \subseteq R$. And it is called* irreflexive *iff $id_A \cap R = \emptyset$.*

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 lemma is left as an easy exercise.

Lemma 4 *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 43-(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 §11), 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 5 *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*.

7.3 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 binary 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 31 and 39 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 $N \subseteq N'$ 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 $N \subseteq N'$ and $G \subseteq G'$; 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 47 *Prove that $f : (N, G) \longrightarrow (N', G')$ is a relation isomorphism iff f is a bijective and full relation homomorphism.*

Exercise 48 *(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 (ii) $f : (A, R) \longrightarrow (B, G)$ is a relation homomorphism iff $R \subseteq f^{-1}(G)$.*

Exercise 49 *(Dedekind-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 Dedekind-Lawvere axiom must be infinite. This tells us that we can use the Dedekind-Lawvere axiom as an alternative to (Inf) to ensure the existence of infinite sets in out set theory. (Hint: Use Exercise 43 (2) to view $(\mathbb{N}, s)$ as a directed graph.)*

[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 23. 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.

Exercise 50 *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.*

7.4 Orders

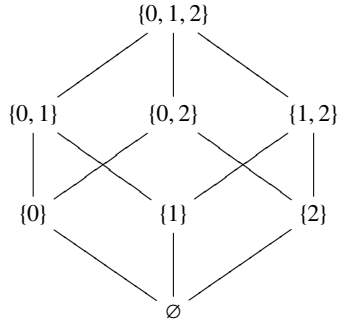
A (strict) *order* on a set A is a relation $< \in \mathcal{P}(A \times A)$ that is both *transitive* and *irreflexive*. We read $a < a'$ as "a is less than a'," or "a is smaller than a'." The inverse relation $<^{-1}$ is denoted $>$, and we read $a > a'$ as "a is greater than a'." An *ordered set* is 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 $>[\{a\}] = \varnothing$. For example, the minimal elements of the poset $(\mathcal{P}(3) - \{\varnothing\}, \subset)$ 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 fellow 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 53 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 51 *Call a relation $R \subseteq A \times A$* asymmetric *iff $R \cap R^{-1} = \varnothing$. 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 52 *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 53 *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 51) $(A \times B, \leq_{A\times B})$ 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 29 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 30 for posets, replacing everywhere the word set by "poset," and the word "function" by "monotonic function."*

Exercise 54 *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 55 *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 56 *(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 $(\mathcal{P}(A), \subseteq)$.*

7.5 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 $lbs(X) = \{\varnothing\}$.

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 $ubs(X) \neq \varnothing$ 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 $\bigwedge X = \varnothing$.

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 6 *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. *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 $ubs(\varnothing) = lbs(\varnothing) = A$. Therefore, we have the, at first somewhat confusing, identities: $\bigvee \varnothing = \bigwedge A = \bot$, and $\bigwedge \varnothing = \bigvee A = \top$. 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 $\bigcap \varnothing = X$, where X is the top element of $(\mathcal{P}(X), \subseteq)$, and of course we have $\bigcup \varnothing = \varnothing$, where $\varnothing$ 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 $\mathcal{P}(1) = \mathcal{P}(\{\varnothing\}) = \{\varnothing, \{\varnothing\}\} = 2$ 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 $0 = \varnothing$ interpreted as *false*, and $1 = \{\varnothing\}$ 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 $y \in X$.

[3] There are many others. The classic textbook in this area is the excellent [5]; 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 [33].

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* [49], 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 $\bot = \text{-}\infty$, then $(\mathbb{R} \cup \{+\infty, \text{-}\infty\}, \leq)$ 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 $\text{-}\infty$ 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 22, you already know what a partial function is. If you missed the fun of Exercise 22, 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 5 *(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 6 *If $(A, \leq)$ is a chain-complete poset with a bottom element $\bot$, 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, $\bot \leq f(\bot)$. Since f is monotonic, we then have a chain $\{f^n(\bot) \mid n \in \mathbb{N}\}$ of the form

$$\bot \leq f(\bot) \leq \ldots \leq f^n(\bot) \leq f^{n+1}(\bot) \leq \ldots$$

Then our desired fixpoint is $\bigvee\{f^n(\bot) \mid n \in \mathbb{N}\}$, because, since f is chain-continuous, we have $f(\bigvee\{f^n(\bot) \mid n \in \mathbb{N}\}) = \bigvee f(\{f^n(\bot) \mid n \in \mathbb{N}\}) = \bigvee\{f^{n+1}(\bot) \mid n \in \mathbb{N}\} = \bigvee\{f^n(\bot) \mid n \in \mathbb{N}\}$. Also, $\bigvee\{f^n(\bot) \mid n \in \mathbb{N}\}$ is the smallest possible fixpoint of f, since if a is any other such fixpoint, then $\bot \leq a$ forces $f^n(\bot) \leq f^n(a) = a$. Therefore, $a \in ubs(\{f^n(\bot) \mid n \in \mathbb{N}\})$, and therefore $\bigvee\{f^n(\bot) \mid n \in \mathbb{N}\} \leq a$, as desired. □

One of the great beauties 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 6 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 for some inputs, and therefore its extensional meaning or *semantics* in general is not a total function, but a partial one.

The importance of Theorem 6 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., [43, 44, 40]). Let me illustrate the basic idea with a simple example. Consider the following recursive function definition to compute the factorial function on natural numbers:

$$\mathit{factorial}(n) = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \cdot \mathit{factorial}(p(n)) \textbf{ fi}$$

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:

$$\delta_{\mathit{factorial}} : [\mathbb{N} \rightharpoonup \mathbb{N}] \longrightarrow [\mathbb{N} \rightharpoonup \mathbb{N}] : f \mapsto \lambda n \in \mathbb{N}.\ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \cdot f(p(n)) \textbf{ fi}.$$

It is not hard to check that $\delta_{\mathit{factorial}}$ 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 $\delta_{\mathit{factorial}}$, that is, the function $\bigvee\{\delta^n_{\mathit{factorial}}(\varnothing) \mid n \in \mathbb{N}\}$, 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

$$\varnothing \subseteq \delta_{\mathit{factorial}}(\varnothing) \subseteq \ldots \subseteq \delta^n_{\mathit{factorial}}(\varnothing) \subseteq \delta^{n+1}_{\mathit{factorial}}(\varnothing) \subseteq \ldots$$

Just by applying the definition of $\delta_{\mathit{factorial}}$ we can immediately see that this is the sequence of partial functions:

$$\varnothing \subseteq \{(0,1)\} \subseteq \ldots \subseteq \{(0,1),(1,2),\ldots,(n,n!)\} \subseteq \{(0,1),(1,2),\ldots,(n,n!),((n+1),(n+1)!)\} \subseteq \ldots$$

Therefore, if we denote by $!\_$ the factorial function, we have, $\bigvee\{\delta^n_{\mathit{factorial}}(\varnothing) \mid n \in \mathbb{N}\} = !\_$, 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 57 *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 58 *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 §7.6 below). Prove that $(\mathit{TransRel}(A), \subseteq)$ and $(\mathit{EquivRel}(A), \subseteq)$ are both complete lattices. Similarly, given a poset $(A, <)$, consider the set $\mathit{Sub}(A, <)$ of all its subposets, that is the set $\mathit{Sub}(A, <) = \{(A', <') \in \mathcal{P}(A) \times \mathcal{P}(A \times A) \mid (A', <') \subseteq (A, <) \wedge \mathit{poset}(A', <')\}$, where, by definition, the predicate $\mathit{poset}(A', <')$ holds iff $(A', <')$ is a poset. Prove that $(\mathit{Sub}(A, <), \subseteq)$ is a complete lattice. (Hint: in each of the three proofs you can cut your work in half using Exercise 57.)*

Exercise 59 *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).*

7.6 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 7 *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 easy to show (exercise) that $a \equiv a'$ iff $[a]_\equiv = [a']_\equiv$. Therefore, equivalence classes are of course pairwise disjoint:

Lemma 5 *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 $A = \emptyset$, then the only equivalence relation is $\emptyset = \emptyset \times \emptyset$, and $\emptyset/\emptyset = \emptyset$, which is a partition of $\bigcup \emptyset = \emptyset$. If $A \neq \emptyset$, obviously $A/\equiv \neq \emptyset$, and $\emptyset \notin A/\equiv$, 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 24. 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 an easy proof, which is left as an exercise.

Lemma 6 *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 7 *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 26), 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 7 to f and the relation $\equiv_f$. This factorization is graphically expressed by the commutative diagram:*

$$\begin{array}{ccc} A & \xrightarrow{\;f\;} & B \\ {\scriptstyle q_{\equiv_f}}\downarrow & & \uparrow{\scriptstyle j^B_{f[A]}} \\ A/{\equiv_f} & \xrightarrow{\;\widehat{f}\;} & f[A] \end{array}$$

Proof. Obviously, by Lemma 7 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 26 (2) and (6), then $\widehat{f} : A/{\equiv_f} \twoheadrightarrow f[A]$ is also injective, and therefore bijective, as desired. □

Exercise 60 *(Exercise 24 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:*

$$\begin{array}{ccc} A & \xrightarrow{\;f\;} & B \\ {\scriptstyle q_{f;f^{-1}}}\downarrow & & \uparrow{\scriptstyle f^{-1};f} \\ A/f; f^{-1} & \xrightarrow{\;\widehat{f}\;} & f[A] \end{array}$$

Exercise 61 *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 62 *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 63 *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 64 *(Extends Exercise 20).*

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 65 *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 66 *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 67 *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 6 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 6.*
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 *pre-order*, and that of *bisimulation*.

Exercise 68 *(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 $\leq_\preceq$ defined by the equivalence $[a]_{\equiv_\preceq} \leq_\preceq [a']_{\equiv_\preceq} \Leftrightarrow a \preceq a'$ is a partial order (in the "less than or equal" sense); and (iii) $q_{\equiv_\preceq} : (A, \preceq) \longrightarrow (A/{\equiv_\preceq}, \leq_\preceq)$ is monotonic and satisfies the following property: for any poset $(B, \leq_B)$ and any monotonic $f : (A, \preceq) \longrightarrow (B, \leq_B)$ there is a unique monotonic $\widehat{f} : (A/{\equiv_\preceq}, \leq_\preceq) \longrightarrow (B, \leq_B)$ such that $f = q_{\equiv_\preceq}; \widehat{f}$.*

Exercise 69 *(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 50–(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 $q_{\equiv_{\mathcal{A}}} : A \longrightarrow\!\!\!\!\!\rightarrow A/\equiv_{\mathcal{A}}$ is a bisimulation $q_{\equiv_{\mathcal{A}}} : \mathcal{A} \longrightarrow\!\!\!\!\!\rightarrow \mathcal{A}/\equiv_{\mathcal{A}}$ (resp. a labeled bisimulation $q_{\equiv_{\mathcal{A}}} : \mathcal{A} \longrightarrow\!\!\!\!\!\rightarrow \mathcal{A}/\equiv_{\mathcal{A}}$).*
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 $\mathcal{A}$.*

7.7 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 an easy 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. In this way, x/y 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 an easy exercise.

Chapter 8

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 8 *(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 $A \ncong B$, 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 $A \ncong B$, 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 9 *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 8. But is every infinite set Dedekind-infinite? The answer to this question will be given in §10.2, Corollary 2.

8.1 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 §**??** 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 7 *(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, $\mathcal{P}^{n+1}(\mathbb{N}) = \mathcal{P}(\mathcal{P}^n(\mathbb{N}))$.

8.2 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 [28].

Theorem 8 *(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 5 (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 70 *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 9

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 can have *repetitions,* like the one for 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.*

9.1 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 $1/s(\_) : \mathbb{N} \longrightarrow\!\!\!\!\!\rightarrow \{1/s(n) \in \mathbb{Q} \mid n \in \mathbb{N}\} : n \mapsto 1/s(n)$, and $2 \cdot \_ : \mathbb{N} \longrightarrow\!\!\!\!\!\rightarrow \dot{2} : n \mapsto 2 \cdot n$. 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 $square : \mathbb{R} \longrightarrow\!\!\!\!\!\rightarrow \mathbb{R}_{\geq 0} : x \mapsto x^2$? 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 $\delta_I : I \longrightarrow\!\!\!\!\!\rightarrow id_I : i \mapsto (i, i)$, 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 $f : I \longrightarrow\!\!\!\!\!\rightarrow T$. 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 10 *(I-indexed set). A surjective function $f : I \longrightarrow\!\!\!\!\!\rightarrow T$ 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 $\widetilde{X} : 1 \longrightarrow\!\!\!\!\!\rightarrow \{X\} : 0 \mapsto X$. 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 $A : I \longrightarrow\!\!\!\!\!\rightarrow A[I]$), 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 [31], 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."

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 $C : 183600 \longrightarrow\!\!\!\!\!\rightarrow C[183600]$, 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

Figure 9.1: C_t for some $t \in 183600$ in the Casablanca movie (from *The Economist, June 14th-20th, pg.89, 2008.*).

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 71 *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 36 and 37, 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

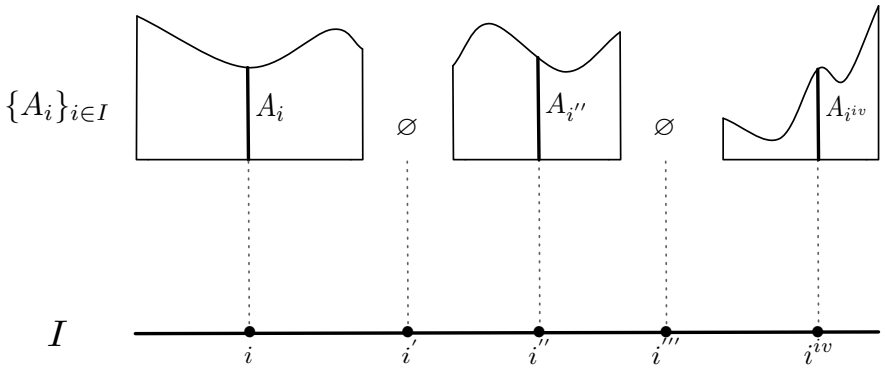


Figure 9.2: Histogram-like display of an I-indexed set $\{A_i\}_{i\in I}$

A_i are spread out above their indices i indicate those cases where $A_i = \varnothing$; 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 lasts 2 minutes and projects during the whole time the single frame $Mao : 3000\times5000 \longrightarrow 2^{16}$ shown in Figure 9.1. That is, since $2\cdot60\cdot30 = 3600$, we have $AVG = \{Mao\}_{t\in3600}$.

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 $X_I : I \longrightarrow\!\!\!\!\!\rightarrow \{X\} : i \mapsto X$. More precisely:

Definition 11 *(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 $X_I : I \longrightarrow\!\!\!\!\!\rightarrow \{X\} : i \mapsto X$. 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 $Array(\mathbb{Q}) : \mathbb{N} \longrightarrow\!\!\!\!\!\rightarrow Array(\mathbb{Q})[\mathbb{N}]$, 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 72 *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 $Array(\mathbb{Q}) : \mathbb{N} \longrightarrow\!\!\!\!\!\rightarrow T$. In general, however, we may be able to describe a "family of sets," in a weaker sense of the term, 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 10 an I-indexed set was *defined* to be a surjective function $A : I \longrightarrow\!\!\!\!\!\rightarrow A[I]$. 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

Figure 9.3: Based on a painting by Andy Warhol in `http://www.warholprints.com/portfolio/Mao.html`.

for each $i \in I$ we have $A_i \in T$, we can always find a surjective function $A : I \longrightarrow\!\!\!\!\!\rightarrow A[I]$ 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

$$\emptyset,\ \mathcal{P}(\emptyset),\ \mathcal{P}(\mathcal{P}(\emptyset)),\ \ldots, \mathcal{P}^n(\emptyset),\ \ldots$$

which we could describe as the family of sets (in this alternative sense, where the surjection is not given) $\{\mathcal{P}^n(\emptyset)\}_{n\in\mathbb{N}}$. It is indeed not at all obvious how to find a set T such that for each $n \in \mathbb{N}$ we have $\mathcal{P}^n(\emptyset) \in T$. An even simpler example is the sequence 0, $\{0\}$ $\{\{0\{\}$ $\ldots$ $\{0\}^n, \ldots$ of Zermelo natural numbers. We know by the axiom of infinity that the set $\mathbb{N}$ of von Neuman natural numbers exists, but is there a T which is the image of the Zermelo sequence? The answer to the question of whether such a set T always exists for an indexed set in this weaker sense is in the affirmative. However, to make this answer precise two additional ideas, which I can only sketch at this point, are needed: (i) a notion of *intensional function* can allow us to precisely characterize a family $\{A_i\}_{i\in I}$ in this weaker sense as a primary notion; and (ii) a new axiom of set theory, the *replacement axiom*, does indeed ensure that a correspoding surjective function $A : I \longrightarrow\!\!\!\!\!\rightarrow A[I]$ always exists for families in this weaker sense.

Exercise 73 *(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 $f : dom(f) \longrightarrow\!\!\!\!\!\rightarrow f[dom(f)]$. Check that all we have said so far can be equivalently formulated by taking I-indexed sets to be untyped functions.*

9.2 Constructing Indexed Sets from other 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 12 *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 $\emptyset_I = \{\emptyset\}_{i \in I}$, that is, the constant I-indexed set associated to $\emptyset$. Obviously, the containment relation $\emptyset_I \subseteq A$ holds true for any I-indexed set $A = \{A_i\}_{i \in I}$.

9.3 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 2 and is left as an exercise.

Lemma 8 *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 $\lambda i \in I.\ \lambda x_i \in A_i.\ x_i \in A_i$.

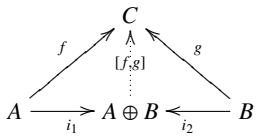
To give a couple of additional examples, illustrating the use of I-indexed functions as parameterized functions for dependent types, consider the following two $\mathbb{N}$-indexed functions from $Array(\mathbb{Q}) = \{[n \to \mathbb{Q}]\}_{n \in \mathbb{N}}$, the dependent type

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 74 *(Generalizes Exercise 30). 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 $h = [f, g]$.*

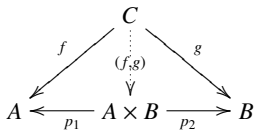
Properties (1)–(3) are compactly expressed by the following commutative diagram of I-indexed functions:



Exercise 75 *(Generalizes Exercise 29). 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:



Chapter 10

From 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 $\bigtimes A$, 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*.

10.1 Some Constructions Associating a Set to an 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 $\bigcup_{i \in I} A_i$, 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 $[n \to \mathbb{Q}] \cap [m \to \mathbb{Q}] = \emptyset$. 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

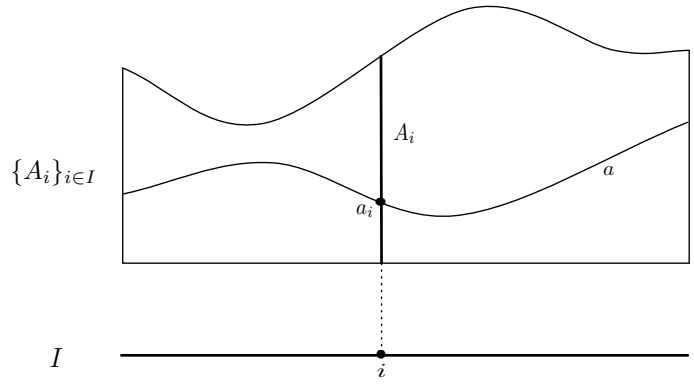
$$\bigtimes_{n \in 2} A_n \longrightarrow A_0 \times A_1 : a \mapsto (a(0), a(1))$$

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 $\bigtimes A = \emptyset$. Indeed, this will always happen as soon as there exists an index $i \in I$ such that $A_i = \emptyset$, 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:

$$\varnothing,\ \ 1,\ \ (1, 1/2),\ \ (1, 1/2, 1/3),\ \ \ldots (1, 1/2, 1/3, \ldots, 1/n), \ldots$$

and which contains the same information as the infinite sequence of rationals

$$1,\ \ 1/2,\ \ 1/3,\ \ 1/4,\ \ \ldots\ \ 1/n,\ \ \ldots$$

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 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

$$\varnothing,\ \ 1,\ \ (1, 1/2),\ \ (1, 1/4, 1/9),\ \ \ldots (1, 1/2^n, 1/3^n, \ldots, 1/n^n), \ldots$$

which does not correspond to any successive approximation of an infinite sequence of rationals. Similarly, the global element

$$\varnothing,\ \ 1,\ \ (2, 2),\ \ (3, 3, 3),\ \ \ldots (n, \overset{n}{\ldots}, n), \ldots$$

does not correspond to any successive approximation of an infinite sequence of rationals either.

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 §9.2, 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 $\mathcal{P}(A)$,

[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.

- 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 76 *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 77 *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 $I \neq \emptyset$, prove the dual equality

$$B \cup (\bigcap A) = \bigcap(B_I \cup A).$$

Exercise 78 *Prove that for any sets I and B the following two set-theoretic equalities hold:*

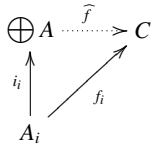
1. $\bigoplus B_I = B \times I$
2. $\bigtimes B_I = [I \to B]$

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 36 is a very special case of (2).

Exercise 79 *(Generalizes Exercise 30). 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 $h = \widehat{f}$.*

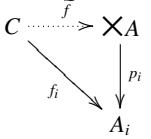
Properties (1)–(2) are graphically expressed by the following commutative diagrams (one for each $i \in I$) of functions:



Exercise 80 *(Generalizes Exercise 29). 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:



Exercise 81 *(Initial and Final Sets Revisited). Show that for the only $\emptyset$-indexed set $id_\emptyset$ we have:*

1. $\bigoplus id_\emptyset = \emptyset$, *and*
2. $\bigtimes id_\emptyset = 1$.

Show, furthermore, that when we specialize Exercises 79 and 80 to the case when $I = \emptyset$, they exactly *mean, respectively, that $\emptyset$ 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 §5.4.*

Exercise 82 *($\bigoplus$ and $\bigtimes$ Are Functorial Constructions) (Generalizes Exercise 32). In a way totally analogous to Exercise 32, 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 79 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 79 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 80 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., $\bigtimes id_A = id_{\bigtimes A}$).

10.2 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:

$$(AC) \qquad (\forall I)(\forall A : I \longrightarrow\!\!\!\!\!\rightarrow A[I])(((\forall i \in I)\ A(i) \neq \emptyset) \ \Rightarrow\ \bigtimes A \neq \emptyset).$$

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 [42]. 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 moral 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 fit of hysteria, 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 84, 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 25, 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 9 AC *is equivalent to every surjective function being a right inverse.*

Proof. To prove the ($\Rightarrow$) part, just notice that $f : A \longrightarrow\!\!\!\!\!\rightarrow B$ 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, $\bigtimes_{b\in B} f^{-1}[\{b\}] \subseteq [B{\rightarrow}A]$ 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 $A_i \neq \emptyset$. This also implies that for each $i \in I$ the set $A_i \times \{i\} \neq \emptyset$. 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 $A_i \times \{i\} \neq \emptyset$, 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, $\bigtimes A \neq \emptyset$, and as a consequence AC holds. □

Note that Theorem 9 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 $g : B \longrightarrow\!\!\!\!\!\rightarrow A$, since by Theorem 9 this ensures the existence of a left inverse (and therefore injective, see Exercise 25) 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 $c : \mathcal{P}(X) - \{\emptyset\} \longrightarrow X$, from the set of its nonempty subsets to X, such that for each $A \in \mathcal{P}(X) - \{\emptyset\}$ 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 10 $AC \Leftrightarrow AC'$.

Proof. For the ($\Rightarrow$) direction, just observe that the identity function $id_{\mathcal{P}(X)-\{\emptyset\}} : \mathcal{P}(X) - \{\emptyset\} \longrightarrow \mathcal{P}(X) - \{\emptyset\}$ is bijective and in particular surjective. Therefore, $id_{\mathcal{P}(X)-\{\emptyset\}}$ is a $(\mathcal{P}(X) - \{\emptyset\})$-indexed set, which we can write in sequence notation as $id_{\mathcal{P}(X)-\{\emptyset\}} = \{A\}_{A\in\mathcal{P}(X)-\{\emptyset\}}$. Of course, since for each $A \in \mathcal{P}(X) - \{\emptyset\}$ the set A is nonempty, by AC the set $\bigtimes id_{\mathcal{P}(X)-\{\emptyset\}} =$

$\bigtimes_{A\in\mathcal{P}(X)-\{\emptyset\}}A$ is nonempty. But since $\bigcup(\mathcal{P}(X)-\{\emptyset\}) = X$, the elements of $\bigtimes_{A\in\mathcal{P}(X)-\{\emptyset\}}A$ are precisely the functions $c : \mathcal{P}(X)-\{\emptyset\} \longrightarrow X$ such that for all $A \in \mathcal{P}(X)-\{\emptyset\}$ 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 $\bigtimes_{A\in\mathcal{P}(X)-\{\emptyset\}}A$, as desired.

For the ($\Leftarrow$) direction, by Theorem 9 it is enough to show that AC' implies that any surjective $f : A \longrightarrow\!\!\!\!\!\rightarrow B$ is a right inverse. But since f is surjective, we have $\{f^{-1}[\{b\}] \mid b \in B\} \subseteq \mathcal{P}(A)-\{\emptyset\}$. Let $c : \mathcal{P}(A)-\{\emptyset\} \longrightarrow A$ be such that for all $C \in \mathcal{P}(A)-\{\emptyset\}$ we have $c(C) \in C$. Then obviously the composed function

$$B \xrightarrow{\{\_\}_B} \mathcal{P}(B) \xrightarrow{f^{-1}[\_]} \mathcal{P}(A)-\{\emptyset\} \xrightarrow{c} A$$

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 11 *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 $\mathcal{P}_\infty(A) \subseteq \mathcal{P}(A)-\{\emptyset\}$ 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 $c : \mathcal{P}(A)-\{\emptyset\} \longrightarrow A$ 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*.

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 [18]; and that of (ii) to Fraenkel (see [47], 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 [47], 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 [47], 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 [27].

Exercise 83 *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 84 *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 85 *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 86 *In the proof of Theorem 11, 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 87 *(The Infinite Hats Puzzle, adapted from [48]). 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 62 useful).

Chapter 11

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.

11.1 Well-Founded Relations

Definition 13 *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 $List(A) = \bigcup_{n \in \mathbb{N}} [n \rightarrow A]$ 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 $x, y, x', y, \ldots$ 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.

[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. For the moment 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 $(+, 1, (*, x, 0))$. In §12.5 I will give a general set-theoretic construction for a set of algebraic expressions and will further discuss this matter.

11.1.1 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 $(A, \varnothing)$ is well-founded. In particular, $(\varnothing, \varnothing)$ is well-founded, and for any well-founded (A, R) there is a unique relation homomorphism $\varnothing : (\varnothing, \varnothing) \longrightarrow (A, R)$.

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 9 *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* construction. Given well-founded relations (A, R) and (B, G), it is easy 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 88 *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 $\{B \to R\}$ defined by the equivalence $f \, \{B \to R\} \, g \;\Leftrightarrow\; (\forall b \in B) \; f(b) \, R \, g(b)$.*

Dually, it is easy 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 89 *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 an easy proof that is left as an exercise.

Lemma 10 *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 is of course a strict poset. Furthermore, any relation (A, R) is well-founded iff (A, R^+) is well-founded.*

As an example, note that $(\mathbb{N}, >) = (\mathbb{N}, p^+)$.

11.2 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 12 *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). □

As an immediate collorary of the above theorem we now obtain an enormously general induction principle, namely, the principle of well-founded induction.

Theorem 13 *(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 $a \in A - B$. □

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 $a_0 \ldots a_{k-1}$ with $k > 1$ we can prove that $P(a_0)$ and $P(a_1 \ldots a_{k-1})$ imply $P(a_0 \ldots a_{k-1})$. 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 $f(t_1, \ldots, t_n)$ we prove that if $P(t_1), \wedge \ \ldots \ \wedge \ P(t_n)$, then $P(f(t_1, \ldots, t_n))$. 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 $a \in A$.

11.3 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.

11.3.1 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 89). 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 $\varnothing$ 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) $rec(f, g, b)(\varnothing) = b$, (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) $map(g)(\varnothing) = \varnothing$, (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, $map(g) = rec(\_ \cdot \_, g, \varnothing)$, 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 $(Exp, \triangleright)$. 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 $f_+, f_-, f_* \in [B \times B \to B]$. 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 $eval(g)(x * (x + y)) = 15$.

11.3.2 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 §11.3.3.

In several of the examples of well-founded recursive function presented in §11.3.1 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 $\bigoplus_{(n,m)\in\mathbb{N}\times\mathbb{N}}[R[\{(n, m)\}]\to\mathbb{N}]$, 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 14 *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 $h : \bigoplus_{a\in A}[R[\{a\}]\to B] \longrightarrow B$, 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 $h : \bigoplus_{a\in A}[R[\{a\}]\to B] \longrightarrow B$ 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 §11.3.3.

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 §11.3.1, 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 Dedekind-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 $h : \bigoplus_{l\in List(A)}[\sqsupset[\{l\}]\to List(\{1\})] \longrightarrow List(\{1\})$, 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 $k > 1$, $h(f, a_0 \dots a_{k-1}) = 1\, f(a_1 \dots a_{k-1})$.

Let $a, b \in A$, 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 90 *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 91 *(Alternative definition of step functions). Given a well-founded relation (A, R) prove that there is a containment $\bigoplus_{a\in A}[R[\{a\}]\to B] \subseteq [A\to B]\times A$. Show that, instead of using a step function $h : \bigoplus_{a\in A}[R[\{a\}]\to B] \longrightarrow B$ to define a well-founded recursive function rec(h), we can always extend such a function h to a function $h' : [A\to B]\times A \longrightarrow B$, so that $h'\restriction_{\bigoplus_{a\in A}[R[\{a\}]\to B]} = h$, 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 $h' : [A\to B]\times A \longrightarrow B$.*

11.3.3 The Well-Founded Recursion Theorem

Let us come back to the three questions raised in §11.3.2: (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 3, for the special case of simple recursion, that is, of well-founded recursion for the well-founded relation $(\mathbb{N}, p)$.

Theorem 14 *(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 an 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 9, 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 an 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 92 *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").*

Part II

Universal Algebra, Equational Logic and Term Rewriting

Chapter 12

Algebras

The traditional realm of algebra was the solution of polynomial equations on numerical domains such as the integers $\mathbb{Z}$ (the so-called Diophantine equations), the rationals $\mathbb{Q}$, the reals $\mathbb{R}$, and the complex numbers $\mathbb{C}$. In the 20th-century, thanks to the set-theoretic notion of mathematical structure pioneered by researchers such as Richard Dedekind, the notion of *Abstract Algebra*, as the study of equationally-defined set-theoretic *structures* with given operations, was vigorously pursued under the leadership of researchers such as Emmy Noether (a classic textbook, gathering the ideas around Emmy Noether and still in use today, is that of van der Waerden [45]).

Abstract Algebra greatly broadened the very notion of algebra in two ways. First, the traditional numerical domains such as $\mathbb{Z}$, $\mathbb{Q}$ $\mathbb{R}$, and $\mathbb{C}$, were now seen as *instances* of more general concepts of *equationally-defined algebraic structure*, which did not depend on any particular representation for their elements, but only on abstract sets of elements, operations on such elements, and equational properties satisfied by such operations. In this way, the integers $\mathbb{Z}$ were seen as an instance of the *ring* algebraic structure, that is, a set R with constants 0 and 1, and with addition $\_ + \_$ and mutiplication $\_ * \_$ operations satisfying the equational axioms of the theory of rings, along with other rings such as the ring $\mathbb{Z}_k$ of the residue classes of integers modulo k, the ring $\mathbb{Z}[x_1, \ldots, x_n]$ of polynomials on n variables, and so on. Likewise, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ were viewed as instances of the *field* structure, that is, a ring F together with a division operator $\_/\_$, so that each nonzero element x has an inverse $1/x$ with $x * (1/x) = 1$, along with other fields such as the fields $\mathbb{Z}_p$, with p prime, the fields of rational functions $\mathbb{Q}(x_1, \ldots, x_n)$, $\mathbb{R}(x_1, \ldots, x_n)$, and $\mathbb{C}(x_1, \ldots, x_n)$ (whose elements are quotients p/q with p, q polynomials and $q \neq 0$), and so on. A second way in which Abstract Algebra broadened the notion of algebra was by considering *other* equationally-defined structures besides rings and fields, such as *monoids*, *groups*, *modules*, *vector spaces*, and so on. This intimately connected algebra with other areas of mathematics such as geometry, analysis and topology in new ways, besides the already well-known connections with geometic figures defined as solutions of polynomal equations (the so-called algebraic varieties, such as algebraic curves or surfaces).

Universal Algebra (the seminal paper is the one by Garett Birkhoff [4]), takes one more step in this line of generalization: why considering only the usual suspects: monoids, groups, rings, fields, modules, and vector spaces? Why not considering *any* algebraic structure defined by an arbitrary collection Σ of function symbols (called a *signature*), and obeying an arbitrary set E of equational axioms? And why not developing algebra in this much more general setting? That is, Universal Algebra is just Abstract Algebra brought to its full generality.

Of course, generalization never stops, so that Universal Algebra itself has been further generalized in various directions. One of them, which we will fully pursue in this Part II and which, as we shall see, has many applications to Computer Science, is from considering a single set of data elements (unsorted algebras) to considering a family of such sets (many-sorted algebras), or a family of such sets but allowing subtype inclusions (order-sorted algebras). Three other, are: (i) replacing the underlying sets by richer structures such as posets, topological spaces, sheaves, or algebraic varieties, leading to notions such as those of an ordered algebra, a topological algebra, or an algebraic structure on a sheaf or on an algebraic variety; for example, an *elliptic curve* is a cubic curve having a commutative group structure; (ii) allowing not only finitary operations but also infinitary ones (we have already seen examples of such algebras with infinitary operations —namely, complete lattices and complete semi-lattices— in §7.5); and (iii) allowing operations to be *partial* functions, leading to the notion of a partial algebra. Order-sorted algebras already provide quite useful support for certain forms of partiality; and their generalization to algebras in *membership equational logic* provides full support for partiality (see [36, 39]).

12.1 Unsorted Σ-Algebras

Unsorted algebras are algebras with a single set of data elements. Since, as we shall see, algebras are the models of theories in equational logic, and equational logic is a sublogic of first-order logic, we should first of all make explicit the *syntax* of equational theories, so that an algebra will then be a set-theoretic *interpretation* of such a syntax. As explained in §2, the syntax of any theory in first-order logic consists of: (i) set of symbols for constants; (ii) a set of function symbols; and (ii) a set of predicate symbols. In equational theories, however, the only predicate symbol is the built-in symbol '=' for equality. Therefore, we only need to specify the syntax of constants and function symbols. This is done by providing a so-called *signature* Σ specifying such symbols.

Definition 15 *An* unsorted signature Σ *is an* $\mathbb{N}$*-indexed family* $\Sigma = \{F_n\}_{n\in\mathbb{N}}$*, where elements* $f \in F_n$ *are called the* n-ary function symbols *of* Σ*. The 0-ary symbols* $c \in F_0$ *are called the* constant symbols*.*

The n-ary function symbols will be interpreted in algebras as operations of n arguments. Recalling the bijection $A \cong [1{\to}A] : a \mapsto \widehat{a}$, explored in detail in Exercise 35, between elements of a set $a \in A$ and functions $\widehat{a} : 1 \longrightarrow A : 0 \mapsto a$, plus the fact that $A^0 = 1$, constants are viewed as *functions of zero arguments*.

For example, the following signatures define the syntax of well-known algebraic structures:

- Σ_{DL}, the signature of *Dedekind-Lawvere natural numbers*, has $F_0 = \{0\}$, $F_1 = \{s\}$, and $F_n = \emptyset$ for $n > 1$.
- Σ_{MON}, the signature of *Monoids*, has $F_0 = \{1\}$, $F_1 = \emptyset$, $F_2 = \{\_ * \_\}$, and $F_n = \emptyset$ for $n > 2$.
- Σ_{GP}, the signature of *Groups*, has $F_0 = \{1\}$, $F_1 = \{(\_)^{-1}\}$, $F_2 = \{\_ * \_\}$, and $F_n = \emptyset$ for $n > 2$.
- Σ_{RNG}, the signature of *Rings*, has $F_0 = \{0, 1\}$, $F_1 = \{\text{-}\_\}$, $F_2 = \{\_ + \_ \_ * \_\}$, and $F_n = \emptyset$ for $n > 2$.
- Σ_{FLD}, the signature of *Fields*, has $F_0 = \{0, 1\}$, $F_1 = \{\text{-}\_, (\_)^{-1}\}$, $F_2 = \{\_ + \_ \_ * \_\}$, and $F_n = \emptyset$ for $n > 2$.

where we either list a symbol as a character or list of characters, as in s, to indicate that s will be displayed with a prefix syntax, so that s applied to x will be denoted $s(x)$, or if we wish to use a convenient "mix-fix" syntax display we follow the useful convention of indicating the argument positions in the syntax of each function symbol by using underbars '_' as for example in $(\_)^{-1}$ to indicate that applied to an argument x will yield $(x)^{-1}$, and $\_ + \_$, to indicate that applied to arguments x and y will yield $x + y$.

A *model* $\mathcal{M}$ of an (unsorted) first-order theory is set M together with an *interpretation* in M for the constants, function symbols, and predicate symbols in the theory's signature. That is, a constant c is interpreted as an element $c_{\mathcal{M}} \in M$, an n-ary function *symbol* f is interpreted as a *function* $f_{\mathcal{M}} : M^n \longrightarrow M$, and an n-ary predicate symbol P is interpreted as an n-ary *relation* $P_{\mathcal{M}} \subseteq M^n$. In particular, if Σ is an unsorted signature as in Definition 15 above with no predicate symbol, such a model is called an *algebra*. The only vagueness left in the above, informal definition is all this talk about "interpretation." Can we get rid of such vagueness? That is, can we make the notion of interpretation precise in set-theoretic terms? Of course!

Definition 16 *Given an unsorted signature* $\Sigma = \{F_n\}_{n\in\mathbb{N}}$*, a* Σ-algebra $\mathcal{A}$ *is an ordered pair* $\mathcal{A} = (A, \_^{\mathcal{A}})$*, where* A *is the set of elements of the algebra, and* $\_^{\mathcal{A}}$ *is called its* structure *or* interpretation function*, and is an* $\mathbb{N}$*-indexed function* $\_^{\mathcal{A}} = \{\_^{\mathcal{A},n} : F_n \longrightarrow [A^n{\to}A]\}_{n\in\mathbb{N}}$.

That is, $\_^{\mathcal{A}}$ interprets each n-ary function *symbol* f as a *function* $f_{\mathcal{A}} : A^n \longrightarrow A$. Implicit in the notion of unsorted Σ-algebra $\mathcal{A} = (A, \_^{\mathcal{A}})$, when viewed as a special case of a model of an unsorted first-order theory, is the intepretation of the built-in equality predicate =, which is interpeted as the *identity relation* on A. That is, $=_{\mathcal{A}} = id_A \subset A^2$.

Note that the notion of Σ-algebra *imposes no particular equational axioms* on an algebra. This is not the role of Σ, which only deals with syntax. Such axioms must be explicitly specified by a set E of equations, so that an *equational theory* is then, as we will see in more detail in §13, a pair (Σ, E), with E a set of equations between expressions built from the syntax of Σ and variables. For example, the theories of groups and rings are, respectively, equational theories of the form (Σ_{GP}, E_{GP}), and (Σ_{RNG}, E_{RNG}), where E_{GP} is the set of equations axiomatizing the theory of groups, and E_{RNG} is the set of equations axiomatizign the theory of rings. Of course, many Σ_{GP}-algebras do *not* satisfy the axioms E_{GP} and therefore are *not* groups; and many Σ_{RNG}-algebras do *not* satisfy the axioms E_{RNG} and therefore are *not* rings. The whole point of universal algebra is that, given a signature Σ, there can be many sets E of equations, giving rise to different theories (Σ, E), (Σ, E'), etc. From this point of view, Σ-algebras are the most general class of algebras possible for the signature Σ, that is, the class of algebras defined by the *empty* set of equations, i.e., by the theory $(\Sigma, \emptyset)$.

In abstract algebra books, except for vector spaces and modules, algebraic structures are typically unsorted. For example, Σ_{MON}-algebras satisfying the equations $E_{MON} = \{x{*}1 = x,\ 1{*}x = x,\ (x{*}y){*}z = x{*}(y{*}z)\}$, are called *monoids*. For A any set, the set $List(A) = \bigcup Array(A)$ of lists of elements of A defined in Section 10.1 is a monoid, with $\_ * \_$ interpreted as string concatenation, i.e., $a_1 \dots a_n * b_1 \dots b_m = a_1 \dots a_n\, b_1 \dots b_m$, and 1 interpreted as the empty string $\emptyset$. Likewise, $\mathbb{N}$ has two obvious *commutative* monoid structures (i.e., monoids satisfying also $x{*}y = y{*}x$): a *multiplicative* one, where 1 is interpreted as $1 \in \mathbb{N}$ and $\_ * \_$ is interpreted as natural number multiplication $\_ * \_ : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, and

an additive one, where 1 is interpreted as $0 \in \mathbb{N}$ and $\_ * \_$ is interpreted as natural number addition $\_ + \_ : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$. Likewise, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ have each a multiplicative, commutative monoid structure, and an additive, commutative monoid structure. But there is more. The just-described additive commutative monoid structures for $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$, can be naturally extended to commutative *group* structures, by interpreting the symbol $(\_)^{-1}$ as the minus operation $-\_$ in, respectively, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$. The additional group theory axiom not already included in the commutative monoid axioms is: $x * (x^{-1}) = 1$, which in its additive interpretation takes the form: $x + (-x) = 0$. Furthermore, Since a commutative ring combines an additive group structure with a multiplicative monoid structure, plus the distributivity axiom $x * (y + z) = (x * y) + (x * z)$, it is easy to check that $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ have all a commutative *ring* structure. Finally, $\mathbb{Q}$-{0}, $\mathbb{R}$-{0}, and $\mathbb{C}$-{0}, are also commutative multiplicative groups, with $*$ interpreted as multiplication, and with the inverse operator $(\_)^{-1}$ interpreted as the multiplicative inverse function $1/\_$ in $\mathbb{Q}$-{0}, $\mathbb{R}$-{0}, and $\mathbb{C}$-{0}. That is, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ are not only commutative rings, but also commutative *fields*[1]

Of course, $\mathbb{N}$ has also a Σ_{DL}-algebra structure, with 0 interpreted as $0 \in \mathbb{N}$, and s as the successor function $s = \lambda n \in \mathbb{N}.\ n \cup \{n\}$. Likewise, the residue classes modulo k, $\mathbb{Z}_k$ have also a Σ_{DL}-algebra structure, with 0 interpreted as $0 \in \mathbb{Z}_k$, and s interpreted as the function: $\lambda n \in \mathbb{Z}_k$. **if** $n <$ (k-1) **then** $n+1$ **else** 0 **fi**. Note that the Σ_{DL}-algebra $\mathbb{N}$ actually satisfies *no* equations (except the trivial one $x = x$), but the Σ_{DL}-algebra $\mathbb{Z}_k$ satisfies the equation $s^k(x) = 0$.

Note, finally, that *there is a substantial abuse of notation* in all the above: one often uses the *same* notation, say, $\mathbb{N}$, to denote *both* the set itself and its intended algebraic structure, and talks of the additive monoid $\mathbb{N}$ or the Σ_{DL}-algebra $\mathbb{N}$. Also, one often uses *symbols* like $0, 1, s, \_ + \_, \_ * \_$, and $-\_$, *to also denote the correspondinresponding functions* in the intended algebra. This is just the customary abuse of notation in mathematics. Whenever notation is abused, by confusing data sets with algebras and function symbols with functions, one should always keep clearly in mind the essential distinction between *sets* and *algebras*; and between the syntactic level of *function symbols* in a signature Σ, which as such are still *uninterpreted*, and the semantic level of actual *functions*, which interpret such function symbols in a given Σ-algebra. To be fully precise, one should use a different notation for each structure, and qualify the function symbols by their interpretation. For example, one could write $\mathbb{N}_{0,+} = (\mathbb{N}, \{0_{\mathbb{N}}, +_{\mathbb{N}}\})$ for the additive monoid structure, and $\mathbb{N}_{0,s} = (\mathbb{N}, \{0_{\mathbb{N}}, s_{\mathbb{N}}\})$ for the Σ_{DL}-algebra structure, on the set $\mathbb{N}$.

Exercise 93 *Let $\Sigma_{DL} = \{F_n\}_{n\in\mathbb{N}}$ be the Dedekind-Lawvere signature, that is, $F_0 = \{0\}$, $F_1 = \{s\}$, and $F_n = \emptyset$ for $n > 1$. And let $A = \{a, b, c\}$. How many different Σ_{DL}-algebras $\mathcal{A} = (A, \_^{\mathcal{A}})$ can be defined on the set A? Can you give a general formula counting, for any finite set B and any unsorted signature $\Sigma = \{F_n\}_{n\in\mathbb{N}}$ with $\bigcup_{n\in\mathbb{N}} F_n$ finite, how many Σ-algebras $\mathcal{B} = (B, \_^{\mathcal{B}})$ there are on B? For example, can your formula* predict *exactly how many such algebras will there be on the above set A if we add to Σ_{DL} binary function symbols $\_ + \_$ and $\_ \times \_$, so that now $F_2 = \{\_ + \_, \_ \times \_\}$?*

12.2 Many-Sorted Σ-Algebras

The passage from unsorted to many-sorted algebras is the analogue in Algebra of the passage in Computer Science from untyped programming languages to typed programming languages. This is *more than an analogy* since, as we shall see later, equational logic can itself be used as a programming language, which can be untyped if we adopt an unsorted equational logic, or can instead be typed using a many-sorted or order-sorted equational logic. In a many-sorted signature we allow a set S of *type names*[2] called *sorts*; and then each function symbol is *typed* in S, by specifying the list of its argument sorts, and the sort of its result sort. Here is the precise definition:

Definition 17 *A* many-sorted signature Σ *is an ordered pair $\Sigma = (S, F)$, where S is called the set of* sorts*, and $F = \{F_{w,s}\}_{(w,s)\in List(S)\times S}$ is the $(List(S) \times S)$-indexed set of its* function symbols*.*

If $w = s_1 \dots s_n$ and $f \in F_{w,s}$, we then display f as $f : s_1 \dots s_n \longrightarrow s$, to indicate that it has argument sorts $w = s_1 \dots s_n$ and result sort s. A *constant* of sort s is a function symbol $c \in F_{\emptyset,s}$, usually denoted $c : nil \longrightarrow s$, since the empty string $\emptyset$ is usually denoted by *nil*.

Note that an unsorted signature is essentially the same thing as a many-sorted signature with a singleton set of sorts, say, $S = \{U\}$, where we think of U as the "universe" sort. The only notational difference between a 1-sorted signature $\Sigma = (\{U\}, F)$ and an unsorted one $\Sigma = \{F_n\}_{n\in\mathbb{N}}$, is that instead of denoting a function symbol f of n arguments as $f \in F_{U\overset{n}{\dots}U,U}$ we leave the universe sort U implicit and write instead $f \in F_n$.

At the level of algebras, unsorted algebras are naturally generalized to many-sorted algebras.

[1] Properly speaking, the theory of fields is not an equational theory, since division by 0 is undefined. We can view a field both as a partial algebra, and as a model of a non-equational first-order theory.

[2] Again, such type names, even if they have suggestive-sounding names such as *Nat*, *Bool*, or *List*, are still *unintepreted names*: they will be interpreted by corresponding sets in a given algebra.

Definition 18 *For $\Sigma = (S, F)$ a many-sorted signature, a* many-sorted Σ-algebra *is a pair $\mathcal{A} = (A, \_^{\mathcal{A}})$, where $A = \{A_s\}_{s \in S}$ is an S-indexed set, and $\_^{\mathcal{A}}$, called the* structure *or* interpretation map*, is a $List(S) \times S$-indexed function*

$$\_^{\mathcal{A}} = \{\_^{\mathcal{A},w,s} : F_{w,s} \longrightarrow [A^w \to A_s]\}_{(w,s) \in List(S) \times S} : f \mapsto f_{\mathcal{A}}$$

where, by defintion, if $w = \mathrm{nil}$, then $A^{\mathrm{nil}} = A^0 = 1$, if $w = \widehat{s} : 1 \longrightarrow S : 0 \mapsto s$, then $A^{\widehat{s}} = A_s$, and if $w = s_1 \ldots s_n$ with $n > 1$, then $A^{s_1 \ldots s_n} = A_{s_1} \times \ldots \times A_{s_n}$.

That is, in $\mathcal{A} = (A, \_^{\mathcal{A}})$, each constant symbol $c : nil \longrightarrow s$ is interpreted as an element $c_{\mathcal{A}} \in A_s$, each $f : s \longrightarrow s'$ as a function $f_{\mathcal{A}} : A_s \longrightarrow A_{s'}$, and each $f : s_1 \ldots s_n \longrightarrow s$ as a function $f_{\mathcal{A}} : A_{s_1} \times \ldots \times A_{s_n} \longrightarrow A_{s'}$. Therefore, a many-sorted algebra $\mathcal{A} = (A, \_^{\mathcal{A}})$ is a *sort-preserving* interpretations of Σ, where each sort s is interpreted as a set A_s, and each function symbol f in F is intepreted as a function $f_{\mathcal{A}}$ respecting its arity sorts and its result sort.

A well-known example of a two-sorted Σ-algebra is a vector space. The signature $\Sigma = (S, F)$ has $S = \{Scalar, Vector\}$, and F has the function symbols: $F = \{0 :\longrightarrow Scalar, 1 :\longrightarrow Scalar, -\_ : Scalar \longrightarrow Scalar, \_ + \_ : Scalar\, Scalar \longrightarrow Scalar, \_ * \_ : Scalar\, Scalar \longrightarrow Scalar, \_ \cdot \_ : Scalar\, Vector \longrightarrow Vector, \vec{0} \longrightarrow Vector, -\_ : Vector \longrightarrow Vector, \_ + \_ : Vector\, Vector \longrightarrow Vector\}$. A *vector space* is exactly a Σ-algebra for this signature such that: (i) the set of scalars is a commutative field, (ii) the set of vectors is a commutative group, and (iii) scalar multiplication satisfies the laws: $a \cdot (v + v') = (a \cdot v) + (a \cdot v')$, $(a + b) \cdot v = (a \cdot v) + (b \cdot v)$, $1 \cdot v = v$, and $(a * b) \cdot v = a \cdot (b \cdot v)$. For example, by choosing $\mathbb{R}$ as the set of scalars with the usual addition, substraction, and multiplication operation on reals, and $\mathbb{R}^3$ as the set of vectors, with vector addition $(a, b, c) + (a', b', c') = (a + a', b + b', c + c')$, and with scalar multiplication $d \cdot (a, b, c) = (d * a, d * b, d * c)$, we get the usual 3-dimensional Euclidean vector space. By relaxing the requirement on scalars to be just a commutative ring instead of a field, and keeping the vectors also as an abelian group with the same equations for scalar multiplication, we get the more general notion of a *module*. For example, by choosing $\mathbb{Z}$ as the set of scalars with the usual addition, substraction, and multiplication operation on reals, and $\mathbb{Z}^3$ as the set of vectors, again with the usual vector addition and scalar multiplication, we get the module of 3-dimensional vectors with integer coordinates.

Other examples of Σ-algebras can be provided by choosing a signature Σ with set of sorts $S = \{List, Nat\}$, and with function symbols $nil : nil \longrightarrow List$, $\_;\_ : List\ List \longrightarrow List$, $0 : nil \longrightarrow Nat$, $s : Nat \longrightarrow Nat$, and $length : List \longrightarrow Nat$. Using the sort names as hints, the most "obvious" Σ-algebras are algebras interpreting *Nat* by $\mathbb{N}$, *List* by $List(A)$, for A a set, and 0 by $0 \in \mathbb{N}$, s by $\lambda n \in \mathbb{N}.\ n \cup \{n\}$, *nil* by $nil \in List(A)$, $\_;\_$ as list concatenation in $List(A)$, and *length* as the function $length : List(A) \longrightarrow \mathbb{N}$ which can be defined by well-founded recursion (see §11.3) by the recursive equations: $length(nil) = 0$, $length(\widehat{a}) = 1$ for each $a \in A$, and $length(a_1\, a_2\, \ldots\, a_n) = s(length(a_2\, \ldots\, a_n))$. However, as already mentioned, it can be misleading to take the hints given by the sort names too literally, since many other interpretations are possible. For example, an equally natural family of Σ-algebras is obtained by interpreting *Nat* by $\mathbb{N}$, *List* by $\mathcal{P}_{fin}(A)$, for A a set, and 0 by $0 \in \mathbb{N}$, s by $\lambda n \in \mathbb{N}.\ n \cup \{n\}$, *nil* by $\emptyset \in \mathcal{P}_{fin}(A)$, $\_;\_$ as set union in $\mathcal{P}_{fin}(A)$, and *length* as the cardinality function $|\_| : \mathcal{P}_{fin}(A) \longrightarrow \mathbb{N}$ which assigns to each finite set $X \in \mathcal{P}_{fin}(A)$ the number of its elements (for more on cardinals and the cardinality function see §§**??** and **??**).

12.3 Order-Sorted Σ-Algebras

In Computer Science one uses many different types of data. Therefore, an unsorted (usually called *unyped*) setting is a bad idea or, if you wish, an inferior technology: it is an unending source of silly bugs that could have been caught if typing had been enforced by a type checker. Often, what amounts to a many-sorted type checking discipline (sometimes under the description of a "simply typed" language) is adopted in many language designs (imperative or declarative, and in either a first-order or a higher-order setting). However, simply typed languages are not expressive enough to deal with partiality issues, such as division by 0, or assigning a type to the head of a list l when l is empty.

An order-sorted typing discipline, in which types are arranged in type hierarchies by subtype inclusion, such as, for example, subtype inclusions $Nat < Int < Rat$ from natural to integers and to rational numbers, and where function symbols can be *subsort overloaded*, so that, for example, addition $\_ + \_$ and multiplication $\_ * \_$ can be defined for the types *Nat*, *Int*, and *Rat* is considerably more flexible than the simply typed discipline offered by many-sorted signatures. Furthermore, an order-sorted typing discipline can quite easily accomodate many often-occurring *partial functions*, and can easily deal with the corresponding type exceptions caused by such functions, which become a nightmare in a simply typed, many-sorted discipline. Common type exceptions of this kind include those caused by expressions such as: $p(0)$, to compute the predecessor of 0, $7/0$, i.e., division by zero, $head(nil)$, i.e., trying to compute the first element of an empty list, and $top(empty)$, i.e., trying to compute the top of an empty stack. The point is that in an order-sorted typing discipline all these partial functions, p, $\_/\_$, *head*, and *top* become *total* in approapriate subtypes, namely, the subtypes *NzNat*, of non-zero natural numbers, *NzRat*, of non-zero rationals, *NeList*, of non-empty lists, and *NeStack* of nonempty stacks.

This typing flexibility is further enhanced by the possibility of giving the benefit of the doubt to expressions that, although in principle meaningful, are detected at parse time to be potentially problematic. For example, it is not clear

at parse time which of the fractions $3/(4 + ((-3) * 3))$ and $3/(4 + ((-2) * 2))$ will have a nonzero denominator, so both should be given the benefit of the doubt, waiting until after they are evaluated to detect type errors. Instead, the expression $3/true$ is utter nonsense and should be immediately rejected by a parser as meaningless. Technically, this extra typing flexibility is achieved as follows. Type names, which we always call *sorts*, are arranged in subsort hierarchies forming a poset $(S, <)$, which, as a graph, may have different *connected components* (see Excercise 67). To give the benefit of the doubt to meaningful expressions such as $3/(4 + ((-3) * 3))$ which cannot be properly typed at parse time (because the parser cannot infer that $(4 + ((-3) * 3)$ has sort *NzInt*), what we can do is: (i) add a fresh new sort, say $\top_{[s]}$, strictly bigger than all sorts in each connected component[3] $[s]$ of the poset of sorts $(S, <)$; and (ii) add for each operator $f : s_1 \dots s_n \longrightarrow s$ in Σ a new, overloaded operator $f : \top_{[s_1]} \dots \top_{[s_n]} \longrightarrow \top[s]$. This is what Maude (see [10]) automatically does, denoting the "kind" $\top_{[s]}$ by just $[s]$. In this way, expressions that would not parse in the original signature Σ, but that are still meaningful, can be given the benefit of the doubt at parse time as expressions at the kind level. That is, both $3/(4 + ((-3) * 3))$ and $3/(4 + ((-2) * 2))$ will be parsed as having kind $[Rat]$. A typing error in the evaluation of an expression now means that the evaluated expression has a kind $[s]$, but has *no sort s* in the original signature Σ. For example, although the expression $3/(4+((-3)*3))$ evaluates to -3/5 of sort *NzRat* without any problems, the expression $3/(4 + ((-2) * 2))$ will evaluate to $3/0$ of kind $[Rat]$. Therefore, having a kind but not having a sort means that the corresponding functional expression is *partial* and is *undefined* for the given arguments. Note that returning a typing error in the form of an evaluated expression of kind $[s]$ gives much more information than returning an "undefined" value such as $\perp$.

Here is the precise definition of an order-sorted signature:

Definition 19 *An* order-sorted signature Σ *is an ordered pair* $\Sigma = ((S, <), F)$ *where: (i)* $(S, <)$ *is a poset, called the poset of* sorts*, and (ii)* (S, F) *is a many-sorted signature.*

That is, an order-sorted signature is just a many-sorted signature to which we have added a partial order relation of subsort inclusion between its sorts. Note that we can recover many-sorted signatures as the special case of order-sorted signatures of the form $\Sigma = ((S, \emptyset), F)$, that is, signatures where sorts are never related by subsort inclusion. This means that order-sorted signatures *generalize* many-sorted signatures, which, in turn, *generalize* unsorted signatures. And, as we shall see shortly, the same happens for algebras: order-sorted algebras *generalize* many-sorted algebras, which, in turn *generalize* unsorted algebras. In the spirit of Universal Algebra, we should develop the relevant concepts and prove everything *at the greatest possible level of generality*. Since any result about order-sorted algebras is *automatically* a result about many-sorted and unsorted algebras as special cases, from now on all further developments will always be carried out at the order-sorted level.

But in this case, the drive for greater generality is not some kind of Bourbaki-like obsession: it is eminently *practical* from the computer science point of view: why using a loser technology such as unsorted signatures and algebras when a much more expressive and useful order-sorted language technology is available? Paradoxically, it is precisely the disregard for practical computer science applications of equational logic and term rewriting to equational specification and programming that lies behind "vanilla flavored," unsorted approaches that cater to theoreticians, but are impractical for real specification and programming.

Note that *overloading of function symbols* was already available in many-sorted signatures: if $\Sigma = (S, F)$ is a many-sorted signature, nothing prevents the *same* function symbol f from having two different typings as $f \in F_{w,s}$, and $f \in F_{w',s'}$, perhaps not just with different argument and result sorts, but also with different numbers of arguments. For example, if we adopted a prefix syntax for a minus symbol - (so that the argument places are not indicated in the syntax), we can have typings - : *Int* $\longrightarrow$ *Int*, and - : *Int Int* $\longrightarrow$ *Int* for unary and binary versions of -. But in a many-sorted setting such different typings are in principle *unrelated* in their semantic intepretation. According to Definion 18, provided we preserve type assignments, if $(w, s) \neq (w', s')$, a many-sorted Σ-algebra $\mathcal{A}$ can associate *any* function with such typing, $f_{\mathcal{A},w,s} \in [A^w, A_s]$ to an $f : w \longrightarrow s$, and a *completely different and unrelated function* $f_{\mathcal{A},w',s'} \in [A^{w'}, A_{s'}]$ to an $f : w' \longrightarrow s'$.

Admittedly, in some particular algebras the fact that we are using the same function symbol may have a certain significance. For example, in the standard intepretation of the function symbols - : *Int* $\longrightarrow$ *Int*, and - : *Int Int* $\longrightarrow$ *Int* in $\mathbb{Z}$, these two different typings of - are related by the equation: $-(x, y) = x + (-y)$. But no relation at all need exist in general. For example, we can have the following two typings of $\_ + \_$ in a many-sorted signature: $\_ + \_$: *Int Int* $\longrightarrow$ *Int*, and $\_ + \_$: *Bool Bool* $\longrightarrow$ *Bool*, and an algebra where *Int* is interpreted as $\mathbb{Z}$, and $\_ + \_$: *Int Int* $\longrightarrow$ *Int* as integer addition, and where *Bool* is interpreted as 2 and $\_ + \_$: *Bool Bool* $\longrightarrow$ *Bool* as the exclusive or function $\_ \boxplus \_$, which is unrelated to integer addition, except for the fact that it can indeed be understood as addition in $\mathbb{Z}_2$. Since in priciple there is no connection between the interpretations that can be given to two diffentent typings of a symbol f in a many sorted signature, such overloading is called *ad-hoc* overloading.

What about function symbol overloading in an order-sorted signature? We can still have ad-hoc overloading in a case like $\_ + \_$: *Int Int* $\longrightarrow$ *Int*, and $\_ + \_$: *Bool Bool* $\longrightarrow$ *Bool*, where there is no semantic relation between the two

[3] As explained in Excercice 67, the connected component of a sort s should properly denoted $[s]_<$. It is the equivalence class associated to s by the smallest equivalence relation $\overline{<}$ generated by $<$. In what follows I will abbreviate $[s]_<$ to just $[s]$.

typings. But we can now also have typings like $\_ + \_ : Nat\ Nat \longrightarrow Nat$, and $\_ + \_ : Rat\ Rat \longrightarrow Rat$, which *are related* to the typing $\_ + \_ : Int\ Int \longrightarrow Int$ in the subsort ordering $Nat < Int < Rat$. These second typings are an instance of what is called *subsort overloading* of function symbols. As we shall see below, the intended *semantic* relation between subsort overloaded function symbols in an order-sorted algebra is that *they should agree on common data*. For example, 2 + 2 should yield 4 as a result, regardless of whether we typed the $\_ + \_$ function symbol to have sort *Nat*, *Int*, or *Rat*. Here is the precise definition.

Definition 20 *Given an order-sorted signature* $\Sigma = ((S, <), F)$*, two typings of a function symbol* f*,* $f : s_1 \dots s_n \longrightarrow s$ *and* $f : s'_1 \dots s'_m \longrightarrow s'$*, are called* subsort-overloaded *iff: (i)* $n = m$*, that is, they have the* same number of arguments*, includingthe case* $n = m = 0$ *where both are constants; and (ii)* corresponding sorts are in the same connect components*, that is,* $[s_1] = [s'_1], \dots, [s_n] = [s'_n]$*, and* $[s] = [s']$*. Otherwise, the typings* $f : s_1 \dots s_n \longrightarrow s$ *and* $f : s'_1 \dots s'_n \longrightarrow s'$ *are called* ad-hoc overloaded*.*

The natural question to ask at this point is: what are the *semantic models* for interpreting order-sorted signatures? Why, of course, order sorted algebras!

Definition 21 *For* $\Sigma = ((S, <), F)$ *an order-sorted signature, an* order-sorted Σ-algebra *is a many-sorted* (S, F)*-algebra* $\mathcal{A} = (A, \_^{\mathcal{A}})$ *such that:*

1. *If* $s < s'$*, then* $A_s \subseteq A_{s'}$*.*
2. *If* $a : nil \longrightarrow s$ *and* $a : nil \longrightarrow s'$ *are two subsort-overloaded typings of a constant* a *in* F*, i.e.,* $[s] = [s']$*, then* $a_{\mathcal{A},nil,s}(\emptyset) = a_{\mathcal{A},nil,s'}(\emptyset)$*, i.e., a is interpreted as the* same *constant (*subsort-overloaded constants coincide*).*
3. *If* $f : s_1 \dots s_n \longrightarrow s$ *and* $f : s'_1 \dots s'_n \longrightarrow s'$ *are two subsort-overloaded typings of* f *in* F *with* $n \geq 1$*, i.e.,* $[s_i] = [s'_i]$*,* $1 \leq i \leq n$*, and* $[s] = [s']$*, then for each* $\vec{a} \in A^{s_1 \dots s_n} \cap A^{s'_1 \dots s'_n}$ *we have* $f_{\mathcal{A},s_1 \dots s_n,s}(\vec{a}) = f_{\mathcal{A},s'_1 \dots s'_n,s'}(\vec{a})$*, that is,* subsort-overloaded operations agree on common data*.*

Of course, conditions (2) and (3) are equivalent to the single condition that for any two subsort-overloaded operators $f : w \longrightarrow s$ and $f : w' \longrightarrow s'$ in F, whenever $\vec{a} \in A^w \cap A^{w'}$, then $f_{\mathcal{A},w,s}(\vec{a}) = f_{\mathcal{A},w',s'}(\vec{a})$, which is spelled out into cases (2) and (3) for the reader's convenience.

Note that conditions (1)–(3) hold trivially for unsorted and many-sorted algebras, that is, the order-sorted algebra notion *subsumes* them as special cases. Therefore, from now on a Σ-algebra will always mean an order-sorted Σ-algebra. Condition (1) is entirely natural: syntactic subsort inclusions $s < s'$ are semantically interpreted as subset inclusions $A_s \subseteq A_{s'}$ on the corresponding data sets. Conditions (2) and (3) take care of the fact that both constants and function symbols can be *subsort-overloaded*, in which case the functions interpreting all these typings *must agree on common data*. For example, we may have an order-sorted signature $\Sigma = ((\{Nat, Int\}, <), F)$, with subsort order $Nat < Int$, and with operations: $F = \{0 :\longrightarrow Nat, 0 :\longrightarrow Int, \_ + \_ : Nat\ Nat \longrightarrow Nat, \_ * \_ : Nat\ Nat \longrightarrow Nat, \_ + \_ : Int\ Int \longrightarrow Int, -\_ : Int \longrightarrow Int, \_ * \_ : Int\ Int \longrightarrow Int\}$. Then we can define an obvious order-sorted Σ-algebra $\mathcal{A}$ by interpreting the sort *Nat* by the set $\mathbb{N}$ of natural numbers, the sort *Int* by the set $\mathbb{Z}$ of integers, but representing numbers in $\mathbb{Z}$ not as equivalence classes, but as $n \in \mathbb{N}$ if n is positive, and as $-n$ if n is a nonzero negative number, which of course gives us the set-theoretic inclusion $\mathbb{N} \subseteq \mathbb{Z}$, as required by (1) for the subsort inclusion $Nat < Int$. Then we interpret 0, +, $*$ as zero, number addition, and number multiplication in, respectively, $\mathbb{N}$ and $\mathbb{Z}$. Then, we have $0_{\mathcal{A},nil,Nat}(\emptyset) = 0_{\mathcal{A},nil,Nat}(\emptyset) = 0$, so that (2) holds, and for any $(n, m) \in \mathbb{N}^2$ we have $+_{\mathcal{A},Nat\,Nat,Nat}(n, m) = +_{\mathcal{A},Int\,Nat,Int}(n, m) = n +_{\mathbb{N}} m = n +_{\mathbb{Z}} m$, and $*_{\mathcal{A},Nat\,Nat,Nat}(n, m) = *_{\mathcal{A},Int\,Nat,Int}(n, m) = n *_{\mathbb{N}} m = n *_{\mathbb{Z}} m$. That is, addition and multiplication of two natural numbers can be performed in either $\mathbb{N}$ or $\mathbb{Z}$ with the *same* result, which is condition (3).

12.4 Terms and Term Algebras

Σ-terms are the algebraic expressions that we can form with the function symbols and constants of a given signature Σ. In an unsorted setting, any function symbol f of n arguments can be applied to any n terms $t_1, \dots, t_n$ to obtain a new term $f(t_1, \dots, t_n)$. For example, we can apply the $\_ + \_$ sign to the arithmetic expressions $2 * 3$ and 7-9 to get the arithmetic expression $(2 * 3) + (7\text{-}9)$, which in prefix form we would write as $+(*(2, 3), \text{-}(7, 9))$. However, in a many-sorted or order-sorted setting, not all terms are meaningful, so that nonsense terms such as $7 + false$ or $nil + (2 * 3)$ should be ruled out. The point is that in a many-sorted or order-sorted setting where S is the set of sorts, we should not define just a *set* of terms, but rather an *S-indexed family of terms*, so that a term t can be typed by its corresponding sort s by a typing relation $t : s$, and the indexed family of terms $T_\Sigma = \{T_{\Sigma,s}\}_{s \in S}$ is such that the elements of $T_{\Sigma,s}$ are exactly those terms t such that $t : s$.

Of course, in an order-sorted setting, a term t *can have several typings*, since if we have $t : s$ and a subsort relation $s < s'$ in the poset of sorts $(S, <)$, then we can also infer that $t : s'$. For example, when we consider subsorts $Nat < Int < Rat$, the term $(2 * 3) + (7\text{-}9)$ obviously has sort *Int*, denoted $(2 * 3) + (7\text{-}9) : Int$, but it has also sort *Rat*,

denoted $(2 * 3) + (7\text{-}9) : Rat$, so that we can further form the fraction term $((2 * 3) + (7\text{-}9))/11 : Rat$. This suggest defining the terms of an order-sorted signature Σ and the typing relation $t : s$ *simultaneously* by the following inductive definition (note that, for simplicity, only the prefix form of a term is considered in this definition):

Definition 22 *Given an order-sorted signature $\Sigma = ((S, <), F)$, 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 $f : s_1 \dots s_n \longrightarrow s$ in F, if $t_1 : s_1, \dots, t_n : s_n$, then $f(t_1, \dots, t_n) : s$. 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 S-indexed family of Σ-terms $T_\Sigma = \{T_{\Sigma,s}\}_{s \in S}$, where, $T_{\Sigma,s}$ denotes the set of Σ-terms of sort s. Note also that, by rule (2), if $s < s'$, then we necessarily have $T_{\Sigma,s} \subseteq T_{\Sigma,s'}$.

Note that the Σ-terms in T_Σ 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 an order-sorted signature $\Sigma = ((S, <), F)$, let us consider a (finite or infinite) S-indexed set of variables $X = \{X_s\}_{s \in S}$, where, we abbreviate $x \in X_s$ by $x : s$, and, to avoid any syntactic confusions, we assume that: (i) if $s \neq s'$, then $X_s \cap X_{s'} = \emptyset$, and (ii) $(\bigcup X) \cap (\bigcup_{s \in S} F_{nil,s}) = \emptyset$. That is, the *names* of the variables are all different, and they are all different from those of the constants in F. Then we can view Σ-terms with variables in X as ground terms on the signature $\Sigma(X) = ((S, <), F(X))$, where, by definition, for any $s \in S$, (i) if $w \neq nil$ then $F(X)_{w,s} = F_{w,s}$, and (ii) $F(X)_{nil,s} = F_{nil,s} \cup X_s$. That is, we just add to Σ the variables in X *as extra constans*. Then a Σ*-term with variables* in X is, by definition, just a (ground) $\Sigma(X)$-term.

Given an order-sorted signature Σ, the S-indexed family of terms $T_\Sigma = \{T_{\Sigma,s}\}_{s \in S}$ can be endowed with a very natural Σ-algebra structure as follows.

Definition 23 *(Term Algebra). For $\Sigma = ((S, <), F)$ an order-sorted signature, the* term algebra *$\mathcal{T}_\Sigma = (T_\Sigma, \_\mathcal{T}_\Sigma)$ is defined by the interpretation map $\_\mathcal{T}_\Sigma$ which maps each $f : s_1 \dots s_n \longrightarrow s$ in F to the function*

$$f_{\mathcal{T}_\Sigma} = \lambda(t_1, \dots, t_n) \in T_{\Sigma,s_1} \times \dots \times T_{\Sigma,s_n}.\ f(t_1, \dots, t_n) \in T_{\Sigma,s}$$

where if $n = 0$, so that $s_1 \dots s_n = nil$, (i.e., when f is a constant), then , by convention, $f(\emptyset) = f$. That is, the operation $f_{\mathcal{T}_\Sigma}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$ acts on a purely syntactic *way on the terms $t_1, \dots, t_n$ to form the new term $f(t_1, \dots, t_n)$.*

Have we seen term algebras before? Yes, of course! They are just the algebras of *data constructors*, such as numbers, lists, or trees (including the *abstract syntax trees* used in parsing) that we use all the time in computer science. What Definition 23 makes explicit is that *data types are algebras*, where the algebraic operations consist on *constructing* bigger data structures out of smaller ones. That is, the operations $f_{\mathcal{T}_\Sigma}$ model exactly the concept of a (free) *data constructor* in data types. They are therefore very useful, since many data types can be defined as term algebras. For example, we can represent the natural numbers as terms in Peano notation with an unsorted signature Σ_{NAT} with a constant symbol 0 and a unary function symbol s. That is, we get a term model of the natural numbers $\mathbb{N}$ as the term algebra $\mathcal{T}_{\Sigma_{NAT}} = (T_{\Sigma NAT}, \_\mathcal{T}_{\Sigma NAT})$, where, say, the successor of the term $s(s(0))$ is precisely the term $s_{\mathcal{T}_{\Sigma NAT}}(s(s(0))) = s(s(s(0)))$. If, instead, we desire a term model of the natural numbers in Zermelo notation, we can use the unsorted signature Σ_{NAT-Z} with a constant $\emptyset$ and a unary operation $\{\_\}$.

Another interesting example is the term algebra of lists with elements in a set A, where Σ_{LIST} is the order-sorted signature with sorts *Elt*, *NeList*, and *List*, subsort inclusions $Elt < NeList < List$, with $F_{nil,Elt} = A$, for A our chosen set of list elements, and with additional operations $nil : nil \longrightarrow List$, and a "cons" operator $\_;\_ : Elt\ NeList \longrightarrow NeList$. Then, the terms of sort *List* in $\mathcal{T}_{\Sigma_{LIST}}$ can be placed in bijective correspondence with the set $List(A)$. For example, if $a, b, c \in A$, then the terms a, and b have sort *Elt*, the terms $a; b$, and $a; b; c; a$ have sort *NeList*, and the term *nil* has sort *List*.

Yet another example is a term model for binary trees with leaf elements in a set A, where Σ_{TREE} is the order-sorted signature with sorts *Leaf*, and *Tree*, subsort inclusion $Leaf < Tree$, with $F_{nil,Leaf} = A$, and with a binary tree constructor $\_ \wedge \_ : Tree\ Tree \longrightarrow Tree$. Then, the terms of sort *Tree* in $\mathcal{T}_{\Sigma_{TREE}}$ are exactly the binary trees with elements from A in their leaves. For example, if $a, b, c \in A$, then the terms a, and b have sort *Leaf*, and $(a \wedge b) \wedge c$, and $(b \wedge c) \wedge (a \wedge (b \wedge b))$ have sort *Tree*.

12.5 A Set-Theoretic Construction of Term Algebras

Strictly speaking, the self-contained nature of these notes has been violated twice. Where? In the introduction of the first-order language of set theory in §2, and in the definition of Σ-terms in Definition 22. That is, the set of formulas in the language of set theory was defined by a context-free grammar, and the set of Σ-terms was inductively defined

using their signature Σ by a syntactic construction equivalent to giving a special type of context-free grammar, called a tree regular grammar (if we allow mix-fix syntax, then sets of terms can be defined and parsed by general context-free grammars). The point, in both cases, is that it is not immediately clear in which sense the *sets* of expressions (formulas or terms) defined by such grammars are definable *within* set theory. That is, such sets have not been justified as set-theoretic constructions. Since inductive definitions of expressions in terms of grammars or of signatures are very intuitive and are used all the time, probably this was not even noticeable, and was not worth remarking on when the violations occurred. Also, heavy use of the first-order language of set theory was essential from almost the very beginning, because of the separation axiom. One needs to mentally bootstrap oneself somehow: self-containment is worth striving for, but may not be fully achievable within reasonable bounds; and if insisted upon to the bitter end it can easily degenerate into pedantry.

Indeed, for the first-order language of set theory, a reasonable answer to give is that such a language provides the *meta-language* in which we carry out our reasoning about sets, so that it would be quite unreasonable to require a bootstrapping process from within set theory to justify its own metalanguage *before* set theory itself has even been defined. Assuming grammars in our meta-language is a very weak assumption, without which one cannot even get off the ground in defining a first-order theory. However, in the case of a term algebra, we are defining an *algebra*, that is, a particular type of set-theoretic structure. Therefore, there is no excuse for not giving a full set-theoretic construction of term algebras, even if a grammar-based description is helpful and illuminating.

The question, therefore, is how to model the inductive set construction of terms in set theory (first-order formulas are a special case, using signatures with sorts *Term*, *Variable*, and *Formula*, with *Variable* < *Term*). Part **??** will provide very general methods for defining sets by transfinite induction which, as shown in §**??**, include the inductive construction of a set of terms as a special case. However, a solution to this problem not needing anything beyond the basic set theory of Part I can be given and is the subject of this section. The key ingredient for this, more elementary solution is implicitly contained in the answer to Exercise 72. If you did not take the hint given there and did not solve Exercise 72, you missed the fun. The answer, of course, is that, given a set A, the sets $[n{\to}A]$ in the $\mathbb{N}$-indexed set $Array(A) = \{[n{\to}A]\}_{n\in\mathbb{N}}$ all belong to the powerset $\mathcal{P}([\mathbb{N}{\to}A])$. In particular, since $List(A) = \bigcup Array(A)$, we have the set-theoretic inclusions

$$[n{\to}A] \subseteq List(A) \subseteq [\mathbb{N}{\to}A].$$

Given an order-sorted signature $\Sigma = ((S, <), F)$, we can model terms as *strings* in an appropriate set of lists, and use Theorem 6 to construct the S-indexed family of sets T_Σ as a fixpoint. To avoid syntactic ambiguities, I will assume that Σ *has no ad-hoc overloading*. This is a very mild assumption, since, if it does, it is very easy to rename the ad-hoc overloaded symbols so that no ad-hoc overloading remains. I will use a slightly more Spartan syntax than prefix syntax, namely, *Polish notation*. Therefore, a term $f(t_1, \ldots, t_n)$ will be displayed as the *string* $f\,\tilde{t}_1 \ldots \tilde{t}_n$, where $\tilde{t}$ is the Polish notation display of t. For example, the arithmetic expression $+(*(2, 3), +(7, 9))$ is displayed in Polish notation as the string $+\ *\ 2\ 3\ +\ 7\ 9$. Since Σ has no ad-hoc overloading, the number of arguments of any function symbol f is uniquely determined, and this makes it easy to unambiguously reconstruct from any term $f\,\tilde{t}_1 \ldots \tilde{t}_n$ in Polish notation the prefix term $f(t_1, \ldots, t_n)$. The set of strings to which all Σ-terms in Polish notation belong is obviously $List(\bigcup F)$. But note that we need to generate not a *set* of terms, but an S-indexed set T_Σ of such terms. This means that in our fixpoint construction of T_Σ we should consider not the powerset $\mathcal{P}(List(\bigcup F))$, but, instead, the complete Boolean algebra (in particular, complete lattice) $[S{\to}\mathcal{P}(List(\bigcup F))]$, where arbitrary sups and infs are defined pointwise, that is, given $H \subseteq [S{\to}\mathcal{P}(List(\bigcup F))]$, we define $\bigvee H = \lambda s \in S.\ \bigcup_{h\in H} h(s)$, and $\bigwedge H = \lambda s \in S.\ \bigcap_{h\in H} h(s)$. The construction of T_Σ is then the minimal fixpoint of the functional

$$T : [S{\to}\mathcal{P}(List(\bigcup F))] \longrightarrow [S{\to}\mathcal{P}(List(\bigcup F))]$$

where, for $h \in [S{\to}\mathcal{P}(List(\bigcup F))]$, T is defined by the lambda expression

$$T = \lambda h.\ \lambda s \in S.\ h(s) \cup \{f\,\tilde{t}_1 \ldots \tilde{t}_n \in List(\bigcup F) \mid (\exists s_1, \ldots, s_n, s' \in S)(f \in F_{s_1 \ldots s_n, s'} \wedge s' \leq s \wedge \tilde{t}_1 \in h(s_1) \wedge \ldots \wedge \tilde{t}_n \in h(s_n))\}.$$

That is, using Theorem 6, we define T_Σ as the minimal fixpoint $T_\Sigma = \bigvee\{T^n(\lambda s \in S.\ \varnothing) \mid n \in \mathbb{N}\}$.

Exercise 94 *Prove in detail that T is a chain-continuous function.*

It may be helpful to unpack the above fixpoint construction for a simple example, such as the unsorted signature Σ_{DL} of the Dedekind-Lawvere natural numbers. In that case, $\bigcup F_{DL} = \{0, s\}$ and, since there is only one sort, T is essentially a functional $T : \mathcal{P}(List(\{0, s\})) \longrightarrow \mathcal{P}(List(\{0, s\}))$, and we have:

$$T(\varnothing) = \{0\},\quad T^2(\varnothing) = \{0, s\,0\},\quad T^3(\varnothing) = \{0, s\,0, s\,s\,0\}, \ldots, T^{n+1}(\varnothing) = \{0, s\,0, \ldots, s\,.^n.\,s\,0\}, \ldots$$

as expected.

So far we have just defined the S-indexed set T_Σ, not the term algebra $\mathcal{T}_\Sigma$. But doing this is quite easy. Just note that for each $f \in \bigcup F$ we have a function $f\_ : List(\bigcup F) \longrightarrow List(\bigcup F) : w \mapsto f\,w$, and for each $n \in \mathbb{N}$ we have the

n-fold append function $append_n : List(\bigcup F)^n \longrightarrow List(\bigcup F)$, with $append_0(\varnothing) = nil$, and $append_{n+1}(w_1, \ldots, w_{n+1}) = w_1\ append_n(w_2, \ldots, w_{n+1})$. Then, for each $f : s_1 \ldots s_n \longrightarrow s$ in F, we define $f_{\mathcal{T}_\Sigma}$ as the function:

$$f_{\mathcal{T}_\Sigma} = \lambda(\tilde{t}_1, \ldots, \tilde{t}_n) \in T_{\Sigma,s_1} \times \ldots \times T_{\Sigma,s_n}.\ f\ append_n(\tilde{t}_1, \ldots, \tilde{t}_n) \in T_{\Sigma,s}.$$

So we are done. There is, however, one nagging question left: how do we *represent* function *symbols*, that is, the elements of the set $\bigcup F$ in set theory? There are essentially two answers. The first answer is that, as I have already mentioned and I further discuss in §**??**, one can use a set theory *with atoms* instead of building all sets *ex nihilo* from the empty set. Then, any desired function symbols can be assumed to be atoms, that is, basic elements of sets which are not themselves sets. The second answer consists in explaining how to represent symbols in pure set theory. There are of course many ways to encode such symbols. The simplest one is perhaps an ASCII-like encoding as the one used in any computer (digital computers, after all, operate on a very restricted fragment of pure set theory!), where the basic characters of an alphabet are placed in bijective correspondence with bit vectors, that is, with the set 2^n for a suitable n, and then identifiers, viewed as character strings, are placed in bijective correspondence with $List(2^n)$. This will work for any Σ with $\bigcup F$ finite or countable, modulo some "padding" to get to the smallest possible 2^n.

Exercise 95 *Assuming that the characters '(' ')' and ',' are either available in a set theory with atoms or suitably encoded in pure set theory, give alternative set-theoretic constructions for T_Σ and $\mathcal{T}_\Sigma$ in which terms are represented in* prefix *notation. Note that we no longer need to require that Σ does not have any ad-hoc overloading.*

12.6 More on Order-Sorted Signatures

Some signatures may be *ambiguous*, so that the same term denotes two different things. This is of course a source of confusion which should be avoided. Consider, for example, a many-sorted signature Σ with set of sorts $S = \{A, B, C, D\}$, a constant symbol $a : nil \longrightarrow A$, and unary function symbols $f : A \longrightarrow B$, $f : A \longrightarrow C$, $g : B \longrightarrow D$, and $g : C \longrightarrow D$. This signature is ambiguous, because the term $g(f(a))$ denotes two different things: (i) the term obtained by first applying $f : A \longrightarrow B$ to a and then applying $g : B \longrightarrow D$; and (ii) the term obtained by first applying $f : A \longrightarrow C$ to a and then $g : C \longrightarrow D$. Of course, in other Σ-algebras we may get completely different results when applying these two difference sequences of operations to the (interpretation of) a. Consider, for example, the Σ-algebra $\mathcal{V} = (V, \_^{\mathcal{V}})$, where $V_A = \{a\}$, $V_B = \{b\}$, $V_C = \{c\}$, and $V_D = \{d, d'\}$, and with $a^{\mathcal{V}} = a$, $f^{\mathcal{V}}_{A,B} = \{(a, b)\}$, $f^{\mathcal{V}}_{A,C} = \{(a, c)\}$, $g^{\mathcal{V}}_{B,D} = \{(b, d)\}$, and $g^{\mathcal{V}}_{C,D} = \{(c, d')\}$. The the term $g(f(a))$ according to interpetation (i) (resp. (ii)) corresponds to the value d (resp. d') in $\mathcal{V}$. That is, $g(f(a))$ denotes two *different* data elements in $\mathcal{V}$ and therefore is intrinsically ambiguous.

Notice that, in the presence of subsort overloading, being unambiguous does *not* forbid the possibility of a term having several sorts. For example, 2 + 2 has sorts *Nat*, *Int*, and *Rat*, using the subsort-overloaded typings $\_ + \_ : Nat\ Nat \longrightarrow Nat$, $\_ + \_ : Int\ Int \longrightarrow Int$, and $\_ + \_ : Rat\ Rat \longrightarrow Rat$. Yet , 2 + 2 is a perfecly unambiguous term. That is, in a many-sorted signature Σ like the one given above, ambiguity will manifest itself by the presence of different *parses* for a term, such as parses (i) and (ii) above. But in an order-sorted signature a term may have different parses without any ambiguity, *provided they are all related in the subsort ordering*. How can we capture this idea? I call an unambiguos order-sorted signature a *sensible* signature. Here is the definition.

Definition 24 *An order-sorted signature $\Sigma = ((S, <), F)$ is* sensible *iff: (i) for any constants of the form $c : nil \longrightarrow s$, and $c : nil \longrightarrow s'$, we must have $[s] = [s']$; and (ii) for any function symbols with same number n of arguments, $n \geq 1$, of the form $f : s_1 \ldots s_n \longrightarrow s$, and $f : s'_1 \ldots s'_n \longrightarrow s'$, if $[s_1] = [s'_1], \ldots, [s_n] = [s'_n]$, then we must have $[s] = [s']$.*

Notice that condition (ii) was the one violated by our ambiguous signature, since for $f : A \longrightarrow B$ and $f : A \longrightarrow C$, we have $[A] = [A] = \{A\}$, but $[B] = \{B\} \neq \{C\} = [C]$. Notice also that *subsort overloaded constants* such as $0 : nil \longrightarrow Nat$ and $0 : nil \longrightarrow Int$ are allowed, but ad-hoc overloaded constants such as $0 : nil \longrightarrow Nat$ and $0 : nil \longrightarrow Bool$ are forbidden by (i). However, ad-hoc overloaded function symbols like $\_ + \_ : Nat\ Nat \longrightarrow Nat$, and $\_ + \_ : Bool\ Bool \longrightarrow Bool$ are allowed, provided (ii) is satisfied or they have different numbers of arguments. Note, finally, that subsort-overloaded symbols automatically satisfy conditions (i)–(ii), so no restriction is placed on them. Since sensibility is such a mild condition and avoids ambiguity, *from now on all signatures will be assumed sensible*.

As already mentioned, in an unambiguous order-sorted signature, a term t may still have different sorts. But which sorts? Here is the answer.

Lemma 11 *If Σ is sensible, for any Σ-term t, if $t : s$ and $t : s'$, then $[s] = [s']$.*

Proof. *We can prove this result by well-founded induction on the immediate subterm relation $f(t_1, \ldots, t_n) \rhd t_i$ (see §11.1). The case for constants follows by condition (i). Suppose now that $f(t_1, \ldots, t_n) : s$, and $f(t_1, \ldots, t_n) : s'$ with $[s] \neq [s']$. This means that we must have $f : s_1 \ldots s_n \longrightarrow s_1$, and $f : s'_1 \ldots s'_n \longrightarrow s'_1$, with $s_1 \leq s$ and $s'_1 \leq s'$, and with $t_1 : s_1, \ldots t_n : s_n$ and $t_1 : s'_1, \ldots t_n : s'_n$. But since Σ is sensible and $[s] \neq [s']$, we must have an i, $1 \leq i \leq n$, such that $[s_i] \neq [s'_i]$. which contradicts the induction hypothesis that $t_i : s_i$ and $t_i : s'_i$ implies $[s_i] = [s'_i]$.* □

Admittedly, since whenever we have $t : s$ and $s < s'$ we also have $t : s'$, a term may have many sorts, although if Σ is sensible all such sorts must belong to a single connected component of $(S, <)$ by Lemma 11. But, in a sense, when we passed from the typing $t : s$ to the typing $t : s'$ using $s < s'$, we *lost information*, since the typing $t : s$ is more precise. For example, $0 : Nat$ is more precise than $0 : Rat$, i.e., gives us more information about 0. This sugest the following question: given a term t, what is the most precise information we can have about its typing? Assuming, as it is always the case in reasonable signatures, that $(S, >)$ is a well-founded set, the most precise type information about t is the set of *minimal elements* in the order $(S, <)$ of the set $sorts(t) = \{s \in S \mid t : s\}$. Obviously, the most informative answer possible about the typing of t is when the set $sorts(t)$ has a *minimum element*, which is the *least sort* possible for t; that is, when the set of minimal elements of $sorts(t)$ is a singleton set. We denote the least sort of t, when it exists, by $ls(t)$. It is easy to give examples of sensible signatures where some terms do no have a least sort. Perhaps the simplest is Σ with $S = \{A, B, C\}$, $A < C$, $B < C$ and a single constant a with $a : nil \longrightarrow A$ and $a : nil \longrightarrow B$. Obviously, $sorts(a) = \{A, B, C\}$, has *two* minimal elements, namely, A and B, so there is no least sort for a. The property that each term t has a least sort is so useful, that it deserves a name of its own.

Definition 25 *A sensible signature Σ is called* preregular *iff for each Σ-term $t \in \bigcup T_{\Sigma(X)}$, the set $sorts(t) = \{s \in S \mid t : s\}$ has a minimum element, denoted $ls(t)$, where X is such that for each $s \in S$, X_s is a sigleton set,*

The reason why we define the notion of preregular signature not for ground terms, but for terms over a set X of variables that has exactly one variable for each sort, is that we want preregularity to be preserved when we extend Σ to $\Sigma(X)$; but this might fail to be the case if for some sorts $s \in S$ the set $T_{\Sigma,s}$ is empty. Consider, for example, a signature Σ with $S = \{A, B, C\}$, $A < B$, $A < C$ and a subsort-overloaded unary function symbol with $f : B \longrightarrow B$ and $f : C \longrightarrow C$. Then we have $\bigcup T_\Sigma = \emptyset$, so, every ground Σ-term trivially has a least sort. However, as soon as we add a variable $x : A$, the term $f(x)$ has two minimal typings, namely, $f(x) : B$, and $f(x) : C$, and therefore has no least sort. The need for considering terms with variables in Definition 25 can be avoided, so that the definition only mentions ground terms, provided Σ has *nonempty sorts*, where, by definition, Σ has nonempty sorts iff for each $s \in S$ we have $T_{\Sigma,s} \neq \emptyset$.

Is there a way to syntactically check that a sensible signature Σ is preregular? Indeed, there is, thanks to the following characterization of preregular signatures in [21], where the notion of preregular signature was first proposed and studied.

Lemma 12 *Let $\Sigma = ((S, <), F)$ be a sensible signature. The the following are equivalent:*

1. *Σ is preregular.*
2. *For each $w \in List(S)$, if $f \in \bigcup_{(w',s),\, w' \geq w,\, s \in S} F_{w',s}$, then the set of sorts $\{s \in S \mid \exists w' \in List(S).\ w' \geq w \wedge f \in F_{w',s}\}$ has a minimum element.*

where, the order $(S, >)$ is extended to the set $List(S)$ in the obvious way: $s_1 \ldots s_n \geq s'_1 \ldots s'_n$ iff $s_i \geq s'_i$, $1 \leq i \leq n$.

Proof. *To see that (1) $\Rightarrow$(2), suppose (2) fails. This means that we have $w \in List(S)$ and $f \in F_{w',s'}$, $f \in F_{w'',s''}$ with $w', w'' \geq w$, and with s' and s'' minimal elements in the set $\{s \in S \mid \exists w' \in List(S).\ w' \geq w \wedge f \in F_{w',s}\}$. Let $w = s_1 \ldots s_n$, $n \geq 0$, then the typings $f(x_1 : s_1, \ldots x_n : s_n) : s'$, and $f(x_1 : s_1, \ldots x_n : s_n) : s''$ are both least possible, so Σ is not preregular.*

The proof that (2) $\Rightarrow$ (1) is by well-founded indution on the immediate subterm relation. In the base case of constants, i.e., when $w = nil$, condition (2) trivially implies that each constant symbol c has a least sort. Consider now a term $f(t_1, \ldots t_n) \in \bigcup T_{\Sigma(X)}$, where, by the induction hypothesis, $ls(t_1) = s_1, \ldots ls(t_n) = s_n$. Then, by (2), the set $\{s \in S \mid \exists w' \in List(S).\ w' \geq s_1 \ldots s_n \wedge f \in F_{w',s}\}$ has a minimum element, say, s'. That is, there is an $f : w' \to s'$ in Σ with $w' \geq s_1 \ldots s_n$ giving the smalles possible typing $f(t_1, \ldots t_n) : s'$, that is, $ls(f(t_1, \ldots t_n)) = s'$, as desired. □

Exercise 96 *Prove that $\Sigma = ((S, <), F)$ is preregular iff $\Sigma(X)$ is preregular for any S-indexed set of variables X satisfying the usual requirement that: (i) if $s \neq s'$, then $X_s \cap X_{s'} = \emptyset$, and (ii) $(\bigcup X) \cap (\bigcup_{s \in S} F_{nil,s}) = \emptyset$.*

We call a sensible signature order-sorted signature $\Sigma = ((S, <), F)$ *topped*, iff each connected component $[s]$ as a top element. If a signature is not topped, it is very easy to extend it to one that is so.

Definition 26 *Let $\Sigma = ((S, <), F)$ be a sensible order-sorted signature. Its* top completion *$\Sigma^\top = ((S^\top, <^\top), F^\top)$ is obtained from Σ by: (i) extending $(S, <)$ to the poset $(S^\top, <^\top)$, where each connected component $[s]$ in $(S, <)$ having a top element is left unchanged, but if $[s]$ lacks a top element, a new one, denoted $\top_{[s]}$, is added above all elements of $[s]$. The operations F are left unchanged, that is, for each $(w, s) \in List(S^\top) \times S^\top$ we have: $F^\top_{w,s} = F_{w,s}$ if $(w, s) \in List(S) \times S$, and $F^\top_{w,s} = \emptyset$ otherwise.*

Exercise 97 *Let $\Sigma = ((S, <), F)$ be an order-sorted signature with top completion $\Sigma^\top = ((S^\top, <^\top), F^\top)$. Prove that:*

1. *If Σ is topped, then $\Sigma^\top = \Sigma$. In particular, $(\Sigma^\top)^\top = \Sigma^\top$.*
2. *For each $s \in S$ we have $T_{\Sigma,s} = T_{\Sigma^\top,s}$.*

3. *If* $[s]$ *lacks a top element in* $(S, <)$*, then,* $T_{\Sigma^\top, \top_{[s]}} = \bigcup_{s' \in [s]} T_{\Sigma, s'}$.

4. *If* Σ *is sensible, so is* $\Sigma^\top$.

5. *If* Σ *is preregular, so is* $\Sigma^\top$.

Finally, as pointed out in §12.3, to give terms the benefit of the doubt it is useful to complete an order-sorted signature Σ by adding a "kind" at the top of ech connected component (even if Σ is topped!), and overloading all function symbols at the kind level. In this way, any term that could eventually evaluate to a term having a sort in the original signature is given the benefit of the doubt. Error or undefined terms, such as $7/0$, are modeled as terms with a kind which do not evaluate to a term with a sort. Let us denote by $\widehat{\Sigma}$ be the signature thus obtained.

Definition 27 *Given an order-sorted signature* $\Sigma = ((S, <), F)$*, its* kind completion $\widehat{\Sigma} = ((\widehat{S}, \widehat{<}), \widehat{F})$ *is obtained from* Σ *by: (i) extending* $(S, <)$ *to the poset* $(\widehat{S}, \widehat{<})$*, where for each connected component* $[s]$ *in* $(S, <)$ *we add a new sort, denoted* $[s]$*, above all sorts in* $[s]$*, and (ii) for any* $f : s_1 \dots s_n \longrightarrow s$ *in* F*,* $(n \geq 0)$*, we add an overloaded version of* f *at the kind level,* $f : [s_1] \dots [s_n] \longrightarrow [s]$ *in* $\widehat{F}$*. We call a signature of the form* $\widehat{\Sigma}$ *a* kind-complete *signature.*

Exercise 98 *Let* $\Sigma = ((S, <), F)$ *be an order-sorted signature with kind completion* $\widehat{\Sigma}$*. Prove that:*

1. *For each* $s \in S$ *we have* $T_{\Sigma, s} = T_{\widehat{\Sigma}, s}$*, that is no new terms are created below the kind level.*

2. *If* Σ *is sensible, so is* $\widehat{\Sigma}$.

3. *If* Σ *is preregular, so is* $\widehat{\Sigma}$.

Chapter 13

Term Rewriting and Equational Logic

13.1 Terms, Equations, and Term Rewriting

Before discussing equations and term rewriting, I introduce some useful concepts and notation for terms, subtems, contexts, and term replacement. Throughout I assume an order-sorted signature $\Sigma = ((S, <), F)$, and its extension $\Sigma(X) = ((S, <), F(X))$ by adding an S-sorted set of variables X. In a Σ-term $f(t_1, \ldots, t_n)$, the $t_1, \ldots, t_n$ are called its *immediate subterms*, denoted $t_i \lhd f(t_1, \ldots, t_n)$, $1 \leq i \leq n$. Note that the inverse relation $\lhd^{-1} = \rhd$ is *well-founded*, and was already described and used in §11.1 for the special case of arithmetic expressions. A term u is called a *subterm* of t iff $t \rhd^* u$, and a *proper subterm* of t iff $t \rhd^+ u$. Note that, as an immediate consequence of Lemma 10 in §11.1, the relation $\rhd^+$ is also well-founded and a strict order. Given a term $t \in \bigcup T_{\Sigma(X)}$, we denote by $vars(t)$ the set of its variables, that is, $vars(t) = \{x \in \bigcup X \mid t \rhd^* x\}$. A term t may contain different occurrences of the same subterm u. 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 new constant [] added[1] to the signature Σ to indicate *where* the subterm u 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,[2] 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, where if $t = C[u]$, then the decomposition of t into the context-subterm pair $(C[], u)$ is succinctly indicated by the more compact notation $C[u]$. For example, we have, among others, the following decompositions of our term $f(b, h(g(a)), g(a))$:

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

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

Definition 28 *(Equations and Equational Theories). Given a sensible order-sorted signature $\Sigma = ((S, <), F)$, a* Σ-equation *is an atomic formula $t = t'$, where $t, t' \in \bigcup T_{\Sigma(X)}$, and where we require that $t = t'$ is* well typed*, in the sense that there are sorts $s, s' \in S$ such that $t \in T_{\Sigma(X),s}$, $t' \in T_{\Sigma(X),s'}$, and $[s] = [s']$. This is obviously less demanding than requiring that t and t' have a common sort; but it is equivalent to such a requirement if Σ is a topped signature. An (unconditional)* equational theory *is then a pair (Σ, E), with Σ a sensible order-sorted signature, and E a set of* Σ*-equations.*

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 $vars(t = t') = \{x_1 : s_1, \ldots, x_n : s_n\}$, where, by definition, $vars(t = t') = vars(t) \cup vars(t')$.

[1] In a kind-complete order-sorted signature Σ with more than one connected component, we should add a new constant [] of kind $[s]$ for each kind $[s]$. To avoid ambiguity, one can qualify the hole constant by its corresponding kind, as $[]_{[s]}$.

[2] Of course, if the hole constant had kind $[s]$, then u should have a sort in $[s]$. In this way, $C[u]$ will always be a well-formed Σ-term. For example, $(3 + 6) * [false]$ is a nonsense term obtained this way, because the whole [] that it replaced has kind [*Natural*], but was replaced by a Boolean term in a different kind. I will assume throughout that this well-kindedness requirement is followed, so that $C[u]$ is always a well-formed Σ-term.

The *proof theory* of equational logic is already familiar to anybody with a high-school education from its use in elementary algebraic manipulations. That is, equational deduction is just the *systematic replacement of equals by equals* using the given equations E. For example, in algebraic manipulations of polynomials we may use ring theory equations such as: (1) $x+y = y+x$, (2) $x*y = y*x$,(3) $(x+y)+z = x+(y+z)$, (4) $x+0 = x$, (5) $x*1 = x$, (6) $x*(y+z) = (x*y)+(x*z)$, and so on, to prove, for example, the polynomial equality $y + (z + (0 + (1 * x)) = (y + z) + x$ by the following sequence of replaments of equals by equals:

$$(\ddagger)\ \ y + (z + [0 + (1 * x)]) = y + (z + [(1 * x) + 0]) = y + (z + [1 * x]) = y + (z + [x * 1]) = [y + (z + x)] = (y + z) + x$$

where I indicate at each point the subterm where an equation is applied by the corresponding term decomposition. The first step has been taken applying equation (1) (in either direction), the second applying equation (4) from left to right, the third applying equation (2) in either direction, the fourth applying equation (5) from left to right, and the last step applying equation (3) from right to left.

We can make the above proof of equality (‡) more informative by giving a name, say *ALG*, to the above set (1)–(6) of equations, and indicating a proof step applying an equation from left to right by $t \rightarrow_{ALG} t'$, a proof step from right to left by $t \leftarrow_{ALG} t'$, and a proof step where we do not indicate the direction (that is, it might be either $t \rightarrow_{ALG} t'$, or $t \leftarrow_{ALG} t'$ —in some special cases, like for equations (1) and (2), it may be *both*— and we do not specify which) by $t \leftrightarrow_{ALG} t'$. With this notation we obtain the more informative proof:

$$y+(z+[0+(1*x)]) \leftrightarrow_{ALG} y+(z+[(1*x)+0]) \rightarrow_{ALG} y+(z+[1*x]) \leftrightarrow_{ALG} y+(z+[x*1]) \rightarrow_{ALG} [y+(z+x)] \leftarrow_{ALG} (y+z)+x.$$

13.1.1 Term Rewriting

Note that certain equations, for example equations (4) and (5) above, have a natural use, when applied from left to right, as *algebraic simplification rules*. This is because their righthand side is clearly simpler, so that applying them systematically from left to right leads to a simpler expression; that is, to a so-called *simplified*, or *reduced* form of the original expression.

Even if an equation's righthand side does not have an obviously simpler appearance, certain directions for applying an equation lead to special syntact forms for fully simplified expressions that are often preferred. For example, applying equation (6) (distributivity) from left to right will lead to the standard way of displaying polynomials as *sums of products*, whereas applying equation (4) (associativity of addition) will right-associate sums, which makes them more readable than some random sum of monomials like $(m_1 + m_2) + (m_3 + (m_4 + (m_5)))$, which gets simplified to $m_1 + (m_2(+(m_3(+m_4 + m_5))))$. Therefore, algebraic simplification produces a special type of equational proofs, called *algebraic simplification proofs*, where equations are *always applied from left to right*. Here is an algebraic simplification proof with equations in *ALG* for a polynomial expression:

$$([x+0]*(y+(z*1)))+x' \rightarrow_{ALG} (x*(y+[z*1]))+x' \rightarrow_{ALG} [x*(y+z)]+x' \rightarrow_{ALG} [((x*y)+(x*z))+x'] \rightarrow_{ALG} (x*y)+((x*z)+x')$$

This process of reduction, i.e., of algebraic simplification is called *term rewriting*. We can make this process explicit by *choosing and orientation* for an equation. That is, we can orient an equation $t = t'$ from left to right as a so-called *rewrite rule* $t \rightarrow t'$, and from right to left as the rewrite rule $t' \rightarrow t$. Of course, some orientations are better than others. For example, the equation $x * 0 = 0$ has a much better orientation as a rewrite rule $x * 0 \rightarrow 0$ than as a rewrite rule $0 \rightarrow x * 0$, not only because 0 is a simpler term than $x * 0$, but also because it has *fewer variables*. The main trouble with using the orientation $0 \rightarrow x * 0$ for algebraic simplification is that we have to *guess* what term should x be instantiated to, since in principle there may be an infinite number of possible instantiations. For this reason, many authors rule out orientations $t \rightarrow t'$ such that $vars(t') \not\subseteq vars(t)$. I will however not impose any restrictions on rewrite rules, although such restrictions will indeed be needed for some applications. Here are the precise definitions of *rewrite rule*, and of a collection of rewrite rules as a *term rewriting system*.

Definition 29 *(Rewrite Rules and Term Rewriting Systems). Given a sensible order-sorted signature $\Sigma = ((S, <), F)$, a* Σ-rewrite rule *is a sequent $t \rightarrow t'$, where $t, t' \in \bigcup T_{\Sigma(X)}$, and where we require that the rule $t \rightarrow t'$ is* well typed*, in the sense that there are sorts $s, s' \in S$ such that $t \in T_{\Sigma(X),s}$, $t' \in T_{\Sigma(X),s'}$, and $[s] = [s']$. This is obviously less demanding than requiring that t and t' have a common sort; but it is equivalent to such a requirement if Σ is a topped signature. A* term rewriting system *(TRS) is then a pair (Σ, R), with Σ a sensible order-sorted signature, and R a set of Σ-rewrite rules.*

We should now formalize the process of term rewriting in a term rewriting system (Σ, R). The first matter that needs to be made precise is the notion of a rule's *instance*. For example, in the last simplification step of the above simplification proof we have used the rule $(x + y) + z \rightarrow x + (y + z)$ by *instantiating* it with the *substitution* $\theta = \{(x, (x * y)), (y, (x * z)), (z, x')\}$, where we think of θ as a function $\theta : \{x, y, z\} \longrightarrow T_{\Sigma_{RNG}(\{x,y,z,x'\})}$. Since we are in a typed, order-sorted setting, variables have sorts, and substitutions must preserve such sorts. The general definition is as follows:

Definition 30 *(Substitutions). Let $\Sigma = ((S, <), F)$ be a sensible order-sorted signature, and let X, Y be S-indexed sets of variables. Then a* substitution *θ is an S-indexed function $\theta = \{\theta_s : X_s \longrightarrow T_{\Sigma(Y),s}\}_{s \in S}$, mapping the variables of X to Σ-terms with variables in Y. If X is* finite*, say, $X = \{x_1 : s_1, \ldots, x_n : s_n\}$, then the S-indexed function θ can be unambiguously specified by a finite set of the form $\theta = \{(x_1, t_1), \ldots, (x_n, t_n)\}$ such that $t_i : s_i$, $1 \leq i \leq n$.*

Given a term $t \in \bigcup T_{\Sigma(X)}$, its instantiation *by θ, denoted $t\theta$, is defined inductively by:*

1. *$x\theta = \theta(x)$, $x \in \bigcup X$*
2. *$a\theta = a$ for each constant $a : nil \longrightarrow s$ in Σ*
3. *$f(u_1, \ldots, u_k)\theta = f(u_1\theta, \ldots, u_k\theta)$.*

For example, for $\theta = \{(x, b), (y, (0 * c))\}$, we have $(x + y)\theta = b + (0 * c)$, and $(y * x)\theta = (0 * c) * b$.

Exercise 99 *(Plugging a subterm into a context as substitution instantiation). Let Σ be a sensible kind-complete signature, and let $[s]$ be one of its kinds. Let Y be an indexed set of variables, and consider the extended indexed set of variables $Y_{[]}$, where $Y_{[],[s]} = Y_{[s]} \uplus \{[]\}$, and $Y_{[],s'} = Y_{s'}$ otherwise. That is, $Y_{[]}$ is obtained from Y just by adding a single new variable $[]$ of kind $[s]$. Call a term $C \in \bigcup T_{\Sigma(Y_{[]})}$ with a single occurrence of the constant $[]$ a* context *with hole of kind $[s]$. We typically write C as $C = C[]$. Let now $u \in T_{\Sigma(Y)[s]}$. Prove that the term $C[u] \in \bigcup T_{\Sigma(Y)}$ obtained by plugging u in the whole $[]$ of $C[]$ is exactly the instantiation $C[]\theta$, for $\theta : Y_{[]} \longrightarrow T_{\Sigma(Y)}$ the substitution such that $\theta([]) = u$, and $\theta(y) = y$ otherwise.*

We are now ready to characterize the term rewriting relation and rewrite proofs.

Definition 31 *(The Rewrite Relation and Rewrite Proofs).*[3] *Let $\Sigma = ((S, <), F)$ be a sensible, kind-complete signature with nonempty sorts, let (Σ, R) be a term rewriting system, and let $Y = \{Y_s\}_{s \in S}$ be an S-indexed set of variables.*[4] *Then an R-*rewrite step *is a pair (u, v), denoted $u \rightarrow_R v$, such that $u \in \bigcup T_{\Sigma(Y)}$ and there is a rewrite rule $t \rightarrow t' \in R$, a substitution $\theta : vars(t \rightarrow t') \longrightarrow T_{\Sigma(Y)}$, and a term decomposition $u = C[t\theta]$ such that $v = C[t'\theta]$, where, by definition, $vars(t \rightarrow t') = vars(t) \cup vars(t')$.*

Since Σ is kind-complete, if $t \rightarrow t' \in R$ and $u = C[t\theta] : [s]$, then we must have $v = C[t'\theta] : [s]$, that is, $\rightarrow_R$ never produces ill-formed terms, i.e., $\rightarrow_R$ is a binary relation $\rightarrow_R \subseteq (\bigcup T_{\Sigma(Y)})^2$. Furthermore, $\rightarrow_R$ is kind-preserving,[5] *i.e., if $u \rightarrow_R v$, $u \rightarrow_R w$, $v : s$, and $w : s'$, then $[s] = [s']$.*

*We denote by $\rightarrow_R^+$ the transitive closure of $\rightarrow_R$, and by $\rightarrow_R^*$ the reflexive-transitive closure of $\rightarrow_R$. A (Σ, R)-*rewrite proof *is, by definition, either a term $t \in \bigcup T_{\Sigma(Y)}$, witnessing a 0-step rewrite proof $t \rightarrow_R^* t$, or a sequence of R-rewrite steps of the form $t_0 \rightarrow_R t_1 \rightarrow_R t_2 \ldots t_{n-1} \rightarrow_R t_n$, with $n \geq 1$, witnessing an n-step rewrite proof $t_0 \rightarrow_R^+ t_n$.*

For example, the algebraic simplification sequence discussed earlier in this Section is a 4-step rewrite proof witnessing $((x+0)*(y+(z*1)))+x' \rightarrow_{ALG}^+ (x*y)+((x*z)+x')$, where $\rightarrow_{ALG}$ is the one-step rewrite relation associated to the set of left-to right rewrite rules $\{t \rightarrow t' \mid t = t' \in ALG\}$, which we shall later denote by $\overrightarrow{ALG} = \{t \rightarrow t' \mid t = t' \in ALG\}$.

13.1.2 Equational Proofs

The notion of an equational proof, that is, a sequence of steps of replacement of equals by equals using equations E, is a trivial instance of the notion of a rewrite proof. Given an equational theory (Σ, E), all we need to do is to consider proofs in the term rewriting system $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E})$, where, by definition, $\overrightarrow{E}$ is the set of left-to-right orientations $\overrightarrow{E} = \{t \rightarrow t' \mid t = t' \in E\}$; and $\overleftarrow{E}$ is the set of right-to-left orientations $\overleftarrow{E} = \{t' \rightarrow t \mid t = t' \in E\}$.

[3] To simplify the exposition I am assuming a sensible order-sorted signature Σ that is *kind-complete* (see Def. 27). This assumptions allows a considerably simpler treatment of term rewriting because, thanks to kind completeness, a rewrite step can never result in a non-well-formed term. Since any order-sorted signature Σ can be naturally extended to a kind-complete one, in practice this assumption does not entail any real loss of generality. I also assume that Σ has nonempty sorts, since this avoids the need for introducing explicit quantifiers.

[4] For generality's sake, I will assume that Y has a countably infinite set of variables Y_s for each $s \in S$, that is, there is a bijective S-indexed function $\alpha : Y \longrightarrow \mathbb{N}_S$; but this requirement is not essential: any S-indexed set of variables Y can be used. That is, the relation $\rightarrow_R$ is in fact parameterized by the chosen family of variables Y.

[5] For $\rightarrow_R$ to be kind-preserving, the assumption that Σ is sensible is crucial, since this ensures Lemma 11. For a simple counterexample, consider Σ with sorts A, B, and C, with corresponding constant symbols a, b, c, and $f : A \longrightarrow B$, $f : A \longrightarrow C$, and R with rules: $f(x : A) \rightarrow b$ and $f(x : A) \rightarrow c$. Then we have $f(a) \rightarrow_R b$, and $f(a) \rightarrow_R c$, but $[B] \neq [C]$.

Definition 32 *(The Equality Relation and Equational Proofs).*[6] *Let* (Σ, E) *be an equational theory with* $\Sigma = ((S, <), F)$ *a sensible, kind-complete signature with nonempty sorts, and* Y *an* S*-indexed set of variables*[7] *Then an* E-equality step *is, by definition, a* $(\overrightarrow{E} \cup \overleftarrow{E})$*-rewrite step* $u \rightarrow_{(\overrightarrow{E} \cup \overleftarrow{E})} v$*, denoted* $u \leftrightarrow_E v$*, where* $u, v \in \bigcup T_{\Sigma(Y)}$*.*

We denote by $\leftrightarrow_E^+$ *the transitive closure of* $\leftrightarrow_E$*; and by* $\leftrightarrow_E^*$ *the reflexive-transitive closure of* $\leftrightarrow_E$*.* $\leftrightarrow_E^*$ *is called the* E-equality relation*, and is often abbreviated to* $=_E$*. It is also called the relation of* equality modulo E*.*

A (Σ, E)-equality proof *is by, definition, either a term* $t \in \bigcup T_{\Sigma(Y)}$*, witnessing a 0-step* E*-equality proof* $t \leftrightarrow_E^* t$*, or a sequence of* E*-equality steps of the form* $t_0 \leftrightarrow_E t_1 \leftrightarrow_E t_2 \ldots t_{n-1} \leftrightarrow_E t_n$*, with* $n \geq 1$*, witnessing an* n*-step equality proof* $t_0 \leftrightarrow_E^+ t_n$*.*

For example, the sequence of equality steps (‡) discussed earlier in §13.1 is a 5-step equality proof witnessing $y + (z + (0 + (1 * x))) \leftrightarrow_{ALG}^+ (y + z) + x$.

Notational Convention. Since in many applications a set of equations E will always be used from left to right, we shall often abbreviate the rewrite relations $\rightarrow_{\overrightarrow{E}}$, $\rightarrow_{\overrightarrow{E}}^+$, and $\rightarrow_{\overrightarrow{E}}^*$, associated to the term rewriting system $(\Sigma, \overrightarrow{E})$ by, respectively, $\rightarrow_E$, $\rightarrow_E^+$, and $\rightarrow_E^*$. Likewise, we will abbreviate the rewrite relations $\rightarrow_{\overleftarrow{E}}$, $\rightarrow_{\overleftarrow{E}}^+$, and $\rightarrow_{\overleftarrow{E}}^*$, associated to the term rewriting system $(\Sigma, \overleftarrow{E})$ by, respectively, $\leftarrow_E$, $\leftarrow_E^+$, and $\leftarrow_E^*$.

Exercise 100 *The theory of groups has an unsorted signature with a constant* 1*, a unary function* $(\_)^{-1}$ *and a binary function* $\_ \circ \_$ *and has the following equations* G*:*

- $x \circ 1 = x$
- $x \circ (y \circ z) = (x \circ y) \circ z$
- $x \circ (x)^{-1} = 1$

Give G*-equality proofs in the sense of Definition 32 for the following theorems of Group Theory:*

- $1 \circ x = x$
- $(x)^{-1} \circ x = 1$
- $(x \circ y)^{-1} = y^{-1} \circ x^{-1}$

(Hint: if you prove any of those theorems, then you can use them as lemmas *to prove some of the others.)*

Exercise 101 *Prove that the* E*-equality relation* $=_E$ *defines an* equivalence relation *on* $\bigcup T_{\Sigma(Y)}$*, and also on* $T_{\Sigma(Y),[s]}$ *for each kind* $[s] \in S/\overleftarrow{<}$*.*

Prove also that, under the assumptions of Definition 32, $=_E$ *does not depend on the set* Y *of variables, chosen to define the equality relation* $=_E$*. That is, given any two* S*-indexed sets of variables* Z, Y *with* $Z \subseteq Y$*, we have in fact two equality relations, which should be notationally distinguished as* $=_E^Z \subseteq (\bigcup T_{\Sigma(Z)})^2$*, and* $=_E^Y \subseteq (\bigcup T_{\Sigma(Y)})^2$*. You are asked to prove that for any terms* $u, v \in \bigcup T_{\Sigma(Z)}$ *we have* $u =_E^Z v$ *iff* $u =_E^Y v$*. The requirement that* Σ *has nonempty sorts is essential for this to be the case, even in the many-sorted case. That is, there are signatures* Σ *and equational theories* (Σ, E) *such that there are* S*-indexed sets of variables* $Z \subseteq Y$ *and terms* $u, v \in \bigcup T_{\Sigma(Z)}$ *such that* $u \neq_E^Z v$*, but* $u =_E^Y v$ *(see, [38, 20]).*

13.2 Term Rewriting and Equational Reasoning Modulo Axioms

Certain equations are *intrinsically problematic* for term rewriting. Consider, for example, the commutativity equations $x + y = y + x$, and $x * y = y * x$. Since the idea of term rewriting is to *simplify* a term to hopefully get a fully simplified, equivalent term, a commutativity equation is intrinsically problematic for two reasons: (i) applying a commutativity equation we do not obtain a simpler term, but only a "mirror image" of the original term; for example, $(x * 7) + (0 * y)$ is rewritten by the commutativity equation to its mirror image $(0 * y) + (x * 7)$; and (ii) even worse, we can *loop* when applying such equations, never reaching a fully simplified term, as in the infinite, alternating sequence

$$(x * 7) + (0 * y) \rightarrow_{ALG} (0 * y) + (x * 7) \rightarrow_{ALG} (x * 7) + (0 * y) \rightarrow_{ALG} (0 * y) + (x * 7) \rightarrow_{ALG} \ldots$$

[6] To simplify the exposition I am assuming a sensible order-sorted signature Σ that is *kind complete* and has *nonempty sorts* (see §12.6). These two assumptions allow a considerably simpler treatment because: (i) no explicit use of universal quantifiers is needed (thanks to the nonempty sorts); and (ii) the replacement of equals by equals can never result in a non-well-formed term (thanks to kind completeness). 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; and if it is not kind complete, care must be taken not to generate non-well-formed terms (see, e.g., [36]). The fact that the presence of empty sorts requires an explicit treatment of universal quantifiers in order to have a *sound* (and of course complete) inference system for equational deduction is well-known, even for many-sorted equational logic, since [38, 20].

[7] Again, with a countably infinite set of variables for each sort $s \in S$ for generality's sake, although this requirement is not essential: any S-indexed set Y can be used.

Of course, this does not block us, humans, from simplifying the above polynomial to $(7 * x)$. But a standard implementation of term rewriting can easily loop when commutativity equations are used as rewrite rules. The solution to this problem, provided by many symbolic algebra systems, and by equational programming languages such as OBJ3 [22], ASF+SDF [46], CafeOBJ [17], ELAN [6], and Maude [10], is to *build in* certain, commonly occurring equational axioms, such as the above commutativity axioms, so that rewriting takes place *modulo* such axioms. For example, we can decompose our equations ALG into a built-in, commutative part $C = \{x + y = y + x, x * y = y * x\}$ and the rest, say, $ALG_0 = \{(x + y) + z = x + (y + z), x + 0 = x, x * 1 = x, x * (y + z) = (x * y) + (x * z)\}$, and then rewrite with the equations in ALG_0 from left to right applying them, not just to a given term t, but to any other term t' which is *provably equal* to t by the equations C, that is, any t' such that $t =_C t'$. This, more powerful rewrite relation is called *rewriting modulo* C, and is denoted $\rightarrow_{ALG_0/C}$. For example, we can simplify the expression $((0 + x) * ((1 * y) + 7)) + z$ to $(x * y) + ((x * 7) + z)$ in just four steps of rewriting with $\rightarrow_{ALG_0/C}$ as follows:

$$((0+x)*((1*y)+7))+z \rightarrow_{ALG_0/C} (x*((1*y)+7))+z \rightarrow_{ALG_0/C} (x*(y+7))+z \rightarrow_{ALG_0/C} ((x*y)+(x*7))+z \rightarrow_{ALG_0/C} (x*y)+((x*7)+z)$$

What rewriting modulo axioms such as C achieves is *raising the level of abstraction* at which we simplify expressions, bringing it closer to the human level. The point is that, since equality modulo a set of axioms like C is an equivalence relation (see Exercise 101), rewriting with $\rightarrow_{ALG_0/C}$ achieves the effect of rewriting not just terms, but *C-equivalence classes of terms*.

But why stopping with commutativity? How about associativity? An associativity equation such as $(x + y) + z = x + (y + z)$ does certainly not have any looping problems; but parentheses around associative operators are a nuisance that can block the application of equations which can "obviously" be applied by humans. For example, the equation $x + -x = 0$ can be applied modulo associativity and commutativity to the expression $((x + y) + z) + -(y + (z + x))$ in *one* step of rewriting *modulo* the following set AC of associativity and commutativity axioms for $\_ + \_$ and $\_ * \_$, $AC = \{x + y = y + x, x * y = y * x, (x + y) + z = x + (y + z), (x * y) * z = x * (y * z)\}$, using from left to right the set of equations $ALG_1 = \{x + 0 = x, x * 1 = x, x * (y + z) = (x * y) + (x * z), x + -x = 0\}$, to the fully simplified form:

$$((x + y) + z) + -(y + (z + x)) \rightarrow_{ALG_1/AC} 0.$$

That is, when rewriting modulo AC: (i) the *order* of the arguments does not matter (because of commutativity, C), and (ii) *parentheses do not matter* (because of associativity, A). In fact, when rewriting modulo associativity (A), or associativity-commutativity (AC), we can disregard parentheses altogether, and write an expression like $((x+y)+z)+x'$ without such parentheses as $x + y + z + x'$, as it is standard practice in mathematics textbooks.

Likewise, we could also build in the unit element axioms $U = \{x + 0 = x, x * 1 = x\}$. Or any combination of C, and/or A, and/or U axioms could be built in. In fact, the idea of building in a set B of equational axioms, so that we rewrite with a set of rules R *modulo* B, is entirely general, and is captured by the notion of a *rewrite theory*.

Definition 33 *Let Σ be a sensible order-sorted signature. A* rewrite theory[8]
is a triple (Σ, B, R), where B is a set of Σ-equations, and R is a set of Σ-rewrite rules.

Rewriting with R *modulo* B can then be formalized as follows.

Definition 34 *(Rewriting and Rewrite Proofs Modulo B). Let (Σ, B, R) be a rewrite theory such that Σ is sensible and kind-complete. Then an R-*rewrite step modulo *B is a pair $(u, v) \in T^2_{\Sigma(Y)}$, denoted $u \rightarrow_{R/B} v$, such that there are terms $u', v' \in T_{\Sigma(Y)}$ with $u =_B u'$, $u' \rightarrow_R v'$, and $v' =_B v$, that is, we have $u =_B u' \rightarrow_R v' =_B v$. We call $\rightarrow_{R/B}$ the* one-step *R-*rewrite relation modulo *B, and denote by $\rightarrow^0_{R/B}$ the relation $=_B$, called the* 0-step *R-*rewrite relation modulo *B, by $\rightarrow^+_{R/B}$ the transitive closure of $\rightarrow_{R/B}$, and by $\rightarrow^*_{R/B}$ the relation[9] $\rightarrow^+_{R/B} \cup =_B$.*

*An R-*rewrite proof modulo *B is either: (i) a pair $(u, v) \in T^2_{\Sigma(Y)}$, with $u =_B v$, witnessing a* 0-step *R-*rewrite modulo *B proof, or (ii) a sequence of R-rewrite steps modulo B of the form $v_0 \rightarrow_{R/B} v_1 \rightarrow_{R/B} v_2 \ldots v_{n-1} \rightarrow_{R/B} v_n$, $n \geq 1$, witnessing an n-step proof $v_0 \rightarrow^+_{R/B} v_n$.*

[8] The traditional use of rewriting modulo axioms has been to reason efficiently, and at a high level of abstraction, in an equational theory $(\Sigma, E \cup B)$ where the equations E are applied from left to right as rewrite rules modulo B. This is the main use we will make of rewrite theories (Σ, B, R) in these notes; that is, to perform simplification modulo B in the equational theory $(\Sigma, B \cup eq(R))$, where $eq(R)$ are the equations associated to the rules R. However, in rewriting logic [35, 37], a rewrite theory (Σ, E, R), has (Σ, E) as its underlying equational theory, but the rules R are *not* interpreted as equations at all, but as *concurrent transitions* in a concurren system, whose states are axiomatized as E-equivalence classes of terms. Futhermore, the equations E in the equational theory (Σ, E) typically decompose as a disjoint union $G \cup B$, where the equations G can be used as simplification rules modulo B; that is, in a simpler rewrite theory $(\Sigma, B, \vec{G})$ used for efficient equational reasoning, as the ones we will consider here. In the Maude language [10], both equational theories of the form $(\Sigma, B \cup G)$, called *functional modules*, and rewrite theories of the general form $(\Sigma, G \cup B, R)$ with a concurrent system semantics, called *system modules* are executed by rewriting with *two* different rewrite relations, namely, $\rightarrow_{G/B}$ for equational simplification, and $\rightarrow_{R/B}$ for concurrent transitions. Maude's functional modules are further discusses in §**??**.

[9] Note that, in general, $\rightarrow^*_{R/B}$ does *not* coincide with the reflexive-transitive closure of $\rightarrow_{R/B}$, but only contains it. It should be thought of as the reflexive-transitive closure of $\rightarrow_{R/B}$ *modulo* B.

A term $u \in \bigcup T_{\Sigma(Y)}$ is said to be in *R/B-normal form* iff $(\nexists u')$. $u \rightarrow_{R/B} u'$, that is, iff u cannot be further rewritten with R modulo B. Also, given a term $t \in \bigcup T_{\Sigma(Y)}$, we say that u is a *R/B-normal form of t* iff $t \rightarrow^*_{R/B} u$, and u is in R/B-normal form, which we abbreviate to $t \rightarrow^!_{R/B} u$.

Exercise 102 *(Deconstructing Rewriting Modulo). What rewriting modulo B accomplishes is to raise the level of abstraction by "building in" B-equality proofs. This suggests that, at the heavy price of losing the higher level of abstraction thus gained, we can "deconstruct" a rewrite theory (Σ, B, R) into a* semantically equivalent *term rewriting system $(\Sigma, R \cup \overrightarrow{B} \cup \overleftarrow{B})$, were we now make explicit each single step of B-equality. But is this true? Prove that if Σ is sensible and kind-complete, then for any two Σ-terms $t, t' \in \bigcup T_{\Sigma(Y)}$ we have $t \rightarrow^*_{R/B} t'$ iff $t \rightarrow^*_{R \cup \overrightarrow{B} \cup \overleftarrow{B}} t'$.*

Exercise 103 *(Equational reasoning modulo axioms). Consider an equational theory of the form $(\Sigma, E \uplus B)$ with Σ sensible and kind-complete. Then we can consider the rewrite theory $(\Sigma, B, \overrightarrow{E} \cup \overleftarrow{E})$, and abbreviate the rewrite relation $\rightarrow_{\overrightarrow{E} \cup \overleftarrow{E}/B}$ to just $\leftrightarrow_{E/B}$. This captures the idea of doing equational reasoning with E modulo the axioms B, which is what we humans do when performing algebraic reasoning, and what smart implementations of equational reasoning do as well. But is it correct? Prove that for any $\Sigma(Y)$-terms t, t' we have the equivalence:*

$$t =_{E \uplus B} t' \quad \Leftrightarrow \quad t \leftrightarrow^*_{E/B} t'.$$

(Hint: use Exercise 102).

13.3 Sort-Decreasingness, Confluence, and Termination

Given a rewrite theory (Σ, B, R), which *executabilty conditions* should be placed in practice on the rules R so that we can effectively use it for equational simplification modulo B in the associated equational theory $(\Sigma, B \cup eq(R))$, in which the rules $t \rightarrow t' \in R$ are now understood as equations $t = t' \in eq(R)$?

As already mentioned for the example of the problematic rule $0 \rightarrow x * 0$ associated to the equation $x * 0 = 0$ when oriented from right to left, the most basic requirement is:

(1) for each $t \rightarrow t' \in R$, *any variable x occuring in t' must also occur in t.*

Otherwise, $t \rightarrow t'$ is problematic as a rewrite rule, 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 rule $0 \rightarrow x*0$ 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 assumption (1), if $u \rightarrow_R v$, then no new variables are introduced in v; that is, we always have $vars(v) \subseteq vars(u)$.

The additional requirement that Σ is *preregular* is also very useful for efficient rewriting. This is because, for a rewrite step $C[t\theta] \rightarrow C[t'\theta]$ with a rule $t \rightarrow t'$ in R to be legal, we need to check that θ is well-sorted; that is, that for each $(x : s, u) \in \theta$ we indeed have $u : s$. But for Σ preregular this just becomes the easy syntactic check $ls(u) \leq s$.

A second important requirement is:

(2) *sort-decreasingness*, that is, for each $t \rightarrow t' \in R$, sort $s \in S$, and substitution θ we should have the implication $t\theta : s \Rightarrow t'\theta : s$.

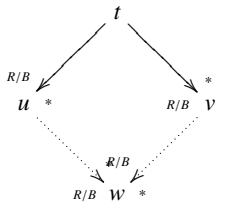
It is then easy to check by well-founded induction on the context C below which a rewrite $C[t\theta] \rightarrow_R C[t'\theta]$ takes place, that under condition (2) the rewrite relation $\rightarrow_R$ *preserves sorts*, that is, if $u \rightarrow_R v$, then $u : s \Rightarrow v : s$. This is of course always the case for unsorted or many-sorted rewriting, but it can fail to hold for order-sorted rewrite theories and, unless the signature Σ is kind-complete, can lead to non-well-formed terms. Consider, for example, a subsort inclusion $Nat < Set$, where Nat has operators 0 and s, and Set has a set union operator $\_ \cup \_$, that is, we view each natural number n as a singleton set. Then the rule $x \rightarrow x \cup x$ with x of sort Set fails to be sort-decreasing, since, for n a variable of sort Nat, the substitution $\theta = \{(x, n)\}$ is such that $n : Nat$, but $n \cup n$ does *not* have sort Nat. This does indeed lead to ill-formed tems, since we then have the rewrite $s[0] \rightarrow s[0 \cup 0]$, where the term $s(0 \cup 0)$ is *not* a well-formed term, unless the signature is kind-complete, so that both s and $\_ \cup \_$ are lifted to the kind $[Set]$. Sort decreasingness is an easily checkable condition, since we do not need to check it on the, in general infinite, set of all substitutions θ: if $\{x_1 : s_1, \ldots, x_n : s_n\} = vars(t \rightarrow t')$, we only need to check it on the, typically finite, set of substitutions of the form $\{(x_1 : s_1, x'_1 : s'_1), \ldots, (x_n : s_n, x'_n : s'_n)\}$ with $s'_i \leq s_i$, $1 \leq i \leq n$, which are called the *sort specializations* [16] of the set of variables $\{x_1 : s_1, \ldots, x_n : s_n\}$. For example, for the above rule $x \rightarrow x \cup x$, the failure was detected for the sort specialization $\{(x : Set, x' : Nat)\}$, which was abbreviated to $\theta = \{(x, n)\}$.

Exercise 104 *Let (Σ, R) be a term-rewriting system with Σ preregular. Prove that the rules R are sort decreasing iff for each sort specialization ρ as defined above and for each $t \rightarrow t'$ in R we have: $ls(t\rho) \geq ls(t'\rho)$.*

To see why things can go wrong when sorts are not preserved by rewriting; that is, when the rules are not sort-decreasing, consider the following simple example. There are two sorts C and D with $C < D$, a constant c of sort C, a constant d of sort D, and a subsort-overloaded unary function $f : C \longrightarrow C$, $f : D \longrightarrow D$. Let $B = \emptyset$ and $R = \{c \to d, f(f(x : C)) \to f(x : C)\}$. Now consider the term $f(f(c))$. Using the second rule we can rewrite it to $f(c)$, which can be further rewritten to $f(d)$ with the first rule. But if we apply the first rule to $f(f(c))$ we get $f(f(d))$ *which cannot be further rewritten!* The point is that the rule $c \to d$ is not sort-decreasing, so the information that c had sort C has been *lost*, and now we are stuck with $f(f(d))$, since we cannot apply the second rule.

A third requirement is one of *determinism*: if a term t is simplified by R modulo B to two different terms u and v, and $u \neq_B v$, then u and v can always be *further simplified* by R modulo E to a common term w. This implies (see Exercise 105 below) that if $t \to^*_{R/B} u$ and $t \to^*_{R/B} v$, and u and v cannot be further simplified by R modulo B, then we must have $u =_B v$. This is the idea of determinism: if rewriting with R modulo B yields a fully simplified answer, then that answer must be *unique* modulo B. That is, the final result of a reduction with the rules R modulo B should *not* depend on the order of evaluation, i.e., on the particular order in which the rewrites have been performed in the rewrite sequence. This is precisely captured by the requirement of *confluence* below. Note that, since the relation $u \to^*_R v$ is just the special case of the relation $u \to^*_{R/B} v$ in which $B = \emptyset$, confluence of R is a special case of confuence of R modulo B.

Definition 35 *Let (Σ, B, R) be a rewrite theory. Then the rules R are called* confluent modulo B *(resp.,* ground confluent modulo B*) iff for each $t \in \bigcup T_{\Sigma(Y)}$ (resp., for each $t \in \bigcup T_\Sigma$), and each pair of rewrites $t \to^*_{R/B} u$, $t \to^*_{R/B} v$, there is a term $w \in \bigcup T_{\Sigma(Y)}$ (resp., $w \in \bigcup T_\Sigma$) such that $u \to^*_{R/B} w$ and $v \to^*_{R/B} w$. This condition can be described diagrammatically as follows (the dashed arrows denote existential quantification):*



Therefore, our third requirement of "determinism" is that:

> (3) the rules R should be *confluent* modulo B (or at least (3′) *ground confluent* modulo B if, as when we use (Σ, B, R) as a declarative program, we are only interested in evaluating ground terms).

Note that, without sort-decreasingness confluence may be hard to get, so conditions (2) and (3) should go together. For example, the already-discussed rewrite rules $R = \{c \to d, f(f(x : C)) \to f(x : C)\}$ with $c : C$, $d : D$, and $C < D$ are *not* confluent, even though the two rules have disjoint function symbols. Indeed, we can rewrite $f(f(c))$ to both $f(d)$ and $f(f(d))$, but these two terms cannot be further rewritten.

Exercise 105 *(Confluence implies uniqueness of normal forms). Let (Σ, B, R) be a rewrite theory. Prove that if (Σ, B, R) is confluent modulo B, then if $t \in \bigcup T_{\Sigma(Y)}$ has a R/B-normal form, then such a normal form is* unique up to B-equality*, that is, v is another R/B-normal form of t iff $u =_B v$.*

*Because of this uniqueness, if (Σ, B, R) is confluent modulo B, we call a R/B-normal form of t, it it exists, a R/B-*canonical form *of t, and denote it, up to B-equality, by $can_{R/B}(t)$.*

Exercise 106 *(Church-Rosser Property). Call two terms $t, t' \in \bigcup T_{\Sigma(Y)}$* joinable *with R modulo B, denoted $t \downarrow_{R/B} t'$, iff $(\exists w \in \bigcup T_{\Sigma(Y)})\ t \to^*_{R/B} w \wedge t' \to^*_{R/B} w$. Prove that if $(\Sigma, E \cup B)$ is an order-sorted equational theory satisfying the requirements in Definition 32, and the rules $\vec{E}$ are confluent modulo B, then the following equivalence, called the* Church-Rosser property*, holds for any two terms $t, t' \in \bigcup T_{\Sigma(Y)}$:*

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

where, as usual, we abbreviate $t \downarrow_{\vec{E}/B} t'$ to just $t \downarrow_{E/B} t'$. (Hint: Use Exercise 103).

It is of course highly desirable that rewriting with R modulo B terminates, so that a final, fully simplified result can be obtained for each term t; so this should be our next requirement.

Definition 36 *Let (Σ, B, R) be a rewrite theory. R is called* terminating *or* strongly normalizing *modulo B (resp.,* ground terminating *or* strongly ground normalizing *modulo B), iff $\to_{R/B}$ is well-founded (resp. $\to_{R/B} \cap (\bigcup T_\Sigma)^2$ is well-founded). We call R* weakly terminating *or* normalizing *modulo B (resp.,* ground weakly terminating *or* ground normalizing *modulo B), iff any $t \in \bigcup T_{\Sigma(Y)}$ (resp., any $t \in \bigcup T_\Sigma$) has a R/B- normal form.*

Therefore, a highly desirable fourth requirement is:

(4) the rules R are terminating modulo B, or at least the weaker requirement (4′) that the rules R are (ground) weakly terminating modulo B.

Exercise 107 *Prove that if* Σ *has non-empty sorts, then given a rewrite theory* (Σ, B, R) *where all rules satisfy condition (1) above,* R *is terminating modulo* B *iff* R *is ground terminating modulo* B.

Exhibit a rewrite theory (Σ, B, R) *satisfying condition (1) where* Σ *has non-empty sorts and* R *is ground weakly terminating modulo* B*, but is not weakly terminating modulo* B.

Exercise 108 *Call a rewrite theory* (Σ, B, R) locally confluent *iff whenever we have* $t \longrightarrow_{R/B} u$ *and* $t \longrightarrow_{R/B} v$*, then* $u \downarrow_{R/B} v$.

- *prove that if* (Σ, B, R) *is terminating and locally confluent, then it is confluent (Hint: use well-founded induction)*
- *exhibit a rewrite theory that is localy confluent but not confluent.*

Essentially, requirements (1)–(4) are all we need of a rewrite theory (Σ, B, R) so that we can effectively use it for equational simplification modulo B in the associated equational theory $(\Sigma, eq(R) \cup B)$. Requirement (1) makes rewriting efficiently implementable, since we do not need to guess the substitution for new variables in the righthand side. Sort decreasingness (requirement (2)) is essential for simplified terms to remain well-formed, so that they do not go up to the kind level if kinds are added, and to avoid that rewrites that were possible before do not get blocked because sort information is lost. Requirement (4) (confluence) is really essential and powerful since, thanks to the Church-Rosser property (see Exercise 106), *equational deduction can be simulated by equational simplification*, in the sense that we have the equivalence $t =_{eq(R) \cup B} t' \;\Leftrightarrow\; t \downarrow_{R/B} t'$. Requirement (4) (termination, or at least weak termination (4')), should not be made into an absolute requirement, but is a very good thing to have.

Of course, even with requirements (1)–(4) all satisfied, unless some further requirements are placed on the equational axioms B so that they can be effectively "built in," the rewrite relation $\rightarrow_{R/B}$ may be hopelessly inefficient (and in general undecidable), since otherwise we may need to actually perform steps of B-equational deduction explicitly. Here are three very useful conditions to require about B. The first one is also well-known for unsorted rewriting, but the remaining two are typical of an order-sorted setting:

- There shoud be a *B-matching algorith*, that is, and algorithm such that, given Σ-terms t and t', gives us a finite, complete[10] set of substitutions θ such that $t\theta =_B t'$, or fails if no such θ exists. If $t\theta =_B t'$ holds, we say that t' *B-matches* the pattern t.
- The variables in the axioms B should all be at the kind level, i.e., of the form $x : [s]$, for $[s]$ a kind in $(S, <)$, so that the equations B apply in their fullest possible generality.
- The equations B should be *B-preregular*, in the sense that, given a B-equivalence class $[t]_B$, the set $\{s \in S \mid t' \in [t]_B \wedge t' : s\}$ has a minimum element, denoted $ls([t]_B)$, that can be effectively computed.

The first condition holds, for example, for B any combination of associativity and/or commutativity and/or identity axioms. For such axioms, the second and third conditions are syntactically checkable, as done in fact in the Maude language [10, 4.4.1 and 22.2.5], where the rules R are also checked and completed so that the first condition achieves in fact the effect of rewriting in B-equivalence classes [10, 4.8].

Exercise 109 *Let* (Σ, B, R) *be a rewrite theory with* Σ *preregular and such that each* $u = v \in B$ *has* $vars(u) = vars(v)$ *and is* sort-preserving *in the precise sense that both* $\overrightarrow{B}$ *and* $\overleftarrow{B}$ *are sort-decreasing (note that this is an easily checkable condition thanks to Exercise 104). Prove the following:*

1. (Σ, B, R) *in B-preregular.*
2. *Whenever* $t \rightarrow_{R/B} t'$*, then* $ls([t]_B) = ls(t) \geq ls(t') = ls([t']_B)$*, that is, sort-decreasingness holds at the level of equivalence classes.*

Also, for B any combination of A and/or C operators, give examples of rewrite theories (Σ, B, R) *with B sort-preserving (with a proof of it) and with B failing to be sort-preserving (with a counterexample).*

Note that the sort-preservation requirement on B is typically too strong in the presence of identity axioms U (a good example of lack of sort preservation is the right identity axiom $x\,;\, nil = x$*, with* $x : List$*,* $\_\,;\,\_$ *an associative list concatenation operator, and with a subsort relation, say,* $Nat < List$*). However, identity axioms can be dealt with using the theory transformation described in [14].*

[10] Complete in the sense that for any other substitution γ not in the set and such that $t\gamma =_B t'$, there is a θ in the set such that for each $x \in vars(t)$ we have $x\gamma =_B x\theta$.

13.4 Canonical Term Algebras

Suppose that we have an equational theory $(\Sigma, E \uplus B)$, where we use the oriented rules $\vec{E}$ to simplify Σ-terms by rewriting with $\vec{E}$ modulo B, that is, with the rewrite theory $(\Sigma, B, \vec{E})$. Suppose, further, that $(\Sigma, B, \vec{E})$ satisfies the executability conditions (1)–(4), or at least the slightly weaker (1)–(2), and (3')–(4'). Then, by termination (or at least weak termination), every term $t \in \bigcup T_\Sigma$ can be simplified to a normal form $can_{E/B}(t)$, so that we have $t \rightarrow!_{E/B} can_{E/B}$. And by confluence (or at least ground confluence), $can_{E/B}(t)$ is *unique* up to B-equality (see Exercise 105). Furthermore, by the Church-Rosser property (see Exercise 106) we have the following extremely useful equivalence for any $t, t' \in \bigcup T_\Sigma$ (which also holds for terms with variables $t, t' \in \bigcup T_{\Sigma(Y)}$ if $(\Sigma, B, \vec{E})$ is confluent):

$$t =_{E \uplus B} t' \quad \Leftrightarrow \quad t \downarrow_{E/B} t' \quad \Leftrightarrow \quad can_{E/B}(t) =_B can_{E/B}(t').$$

This is indeed very powerful! In order for us to *know* if two terms t, t' are provably equal in the theory $(\Sigma, E \uplus B)$ all we need to do is to *reduce them to canonical form* with $\rightarrow_{E/B}$ and perform the equality test $can_{E/B}(t) =_B can_{E/B}(t')$, which if B has a B-matching algorithm is a decidable test. Of course, for the special case $B = \emptyset$, this becomes a test for *syntactic equality* $can_E(t) = can_E(t')$.

This suggests considering the terms in E/B-canonical form as the *values of an algebra*. Let us consider a simple example, namely, an unsorted signature Σ with a constant 0, a unary successor function s, and a binary addition function $\_ + \_$, and the following set E of equations: $E = \{x + 0 = x, x + s(y) = s(x + y)\}$. It is easy to check that the term rewriting system $(\Sigma, \vec{E})$ is confluent and terminating. It is also easy to check that the set of ground terms in $\vec{E}$-canonical form is the set $Can_{\Sigma/E} = \{0, s(0), s(s(0)), \ldots, s^n(0), \ldots\}$, that is the natural numbers in Peano notation. This is certainly a set of values, but for which algebra? Well, for each operation on such values, we can *agree* that the result of the operation is, by definition, its *canonical form*. That is, we can define a Σ-algebra $C_{\Sigma/E} = (Can_{\Sigma/E}, \_{C_{\Sigma/E}})$ as follows:

1. $0_{C_{\Sigma/E}} = can_E(0) = 0$.
2. for each $t \in Can_{\Sigma/E}$, $s_{C_{\Sigma/E}}(t) = can_E(s(t)) = s(t)$.
3. for each $(t, t') \in Can^2_{\Sigma/E}$, $t +_{C_{\Sigma/E}} t' = can_E(t + t')$.

Obviously, in cases (1) and (2), terms like 0, or $s^{n+1}(0)$ are already in canonical form, so the results are just purely syntactic, like in a term algebra. But case (3) is different. Let us try some examples:

- $0 +_{C_{\Sigma/E}} 0 = 0$
- $s(0) +_{C_{\Sigma/E}} s(0) = s(s(0))$
- $s(s(0)) +_{C_{\Sigma/E}} s(s(0)) = s(s(s(s(0))))$
- $s(s(s(0))) +_{C_{\Sigma/E}} s(s(0)) = s(s(s(s(s(0)))))$

So, it turns out that the function $\_ +_{C_{\Sigma/E}} \_$ is just the *addition function on natural numbers!* What could be more natural! So, the algebra $C_{\Sigma/E}$ of terms in E-canonical form is just the *algebra of natural numbers* with the *standard* intepretation for the symbols 0, s, and $\_ + \_$.

Here is the general definition, under the weakest possible assumptions.

Definition 37 *(Canonical Term Algebra). Let $(\Sigma, E \uplus B)$ be an equational theory satisfying conditions (1)–(2), which is also (3') ground confluent, and (4') weakly terminating. Then the S-indexed set $Can_{\Sigma/E,B} = \{Can_{\Sigma/E,B,s}\}_{s \in S}$, where for each $s \in S$ we define $Can_{\Sigma/E,B,s} = \{[can_{E/B}(t)]_B \in T_{\Sigma,[s]}/=_B \mid t \in T_{\Sigma,[s]} \wedge \exists t' \in [can_{E/B}(t)]_B, t' : s\}$, can be given a Σ-algebra structure called the* canonical term algebra *associated to the theory $(\Sigma, E \uplus B)$ and denoted $C_{\Sigma/E,B} = (Can_{\Sigma/E,B}, \_{C_{\Sigma/E,B}})$, where the structure map $\_{C_{\Sigma/E,B}}$ assigns to each $f : w \longrightarrow s$ in Σ the function $f_{C_{\Sigma/E,B}} : Can^w_{\Sigma/E,B} \longrightarrow Can_{\Sigma/E,B,s}$, which is defined:*

- *for $w = nil$, by the identity $f_{C_{\Sigma/E,B}}(\emptyset) = can_{E/B}(f)$, and*
- *for $w = s_1 \ldots s_n$, $n \geq 1$, by the function*

$$f_{C_{\Sigma/E,B}} = \lambda([t_1]_B, \ldots, [t_n]_B) \in Can_{\Sigma/E,B,s_1} \times \ldots \times Can_{\Sigma/E,B,s_n}.\ [can_{E/B}(f(t_1, \ldots, t_n))]_B.$$

Exercise 110 *Prove that under the assumptions of Definition 37 the function $f_{C_{\Sigma/E,B}} = \lambda([t_1]_B, \ldots, [t_n]_B) \in Can_{\Sigma/E,B,s_1} \times \ldots \times Can_{\Sigma/E,B,s_n}.\ [can_{E/B}(f(t'_1, \ldots, t'_n))]_B$ is well-defined, that is, it does not depend on the choice of representatives $t_i \in [t_i]_B$, $1 \leq i \leq n$.*

13.5 Sufficient Completeness

Consider again our equations $E = \{x + 0 = x, x + s(y) = s(x + y)\}$ on the unsorted signature Σ with symbols $0, s$ and $\_ + \_$. And observe the interesting point that its set $Can_{\Sigma/E}$ of canonical forms is precisely the set T_{DL} of terms in the Dedekind-Lawvere signature with symbols 0 and s. That is, the *addition symbol has completely disappeared!* This is as it should be, since the equations $E = \{x + 0 = x, x + s(y) = s(x + y)\}$ provide a *complete* definition of the addition function on natural numbers. Note that we obviously have a strict inclusion $\Sigma_{DL} \subset \Sigma$.

This is a more general fact, not just about numbers, but about any equational theory $(\Sigma, E \uplus B)$ satisfying conditions (1)–(2) and (3′)–(4′), where some operations in a subsignature $\Omega \subseteq \Sigma$ (in our example $\Sigma_{DL} \subset \Sigma$) are used as *data constructors*, and the remaning operations in Σ-Ω are viewed as *functions* operating on data built with the data constructors Ω and returning as result another data value built with the constructors Ω. The *definition* of the function associated to a function symbol $f \in \Sigma$-Ω is given by the equations E modulo B or, more precisely, by the *function* $f_{C_{\Sigma/E,B}}$ associated to f in the canonical term algebra $C_{\Sigma/E,B}$. In our example, we had Σ-$\Sigma_{DL} = \{\_ + \_\}$, and the function $\_ +_{C_{\Sigma/E}} \_$ was just the addition function on natural numbers.

This is useful, because we can then ask the question: have we given *enough equations* to *fully define* all functions in Σ-Ω? 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 canonical form has only constructor symbols. This would fail, for example, if we had forgotten to define the case of adding 0 to a number, and had tried to define addition by the single equation $x + s(y) = s(x + y)$. Then we would have terms in canonical form like $0 + 0$, $s(0) + 0$, $s((s(0) + 0) + 0)$, and so on, so that the + symbol would no longer disappear. For complex specifications, forgetting some corner case like this when defining a function is not uncommon. Such forgetfulness is called a *failure of sufficient completeness*, in the sense that the function f in question has not been sufficiently defined, because some equations are missing. And the failure is detected precisely by the presence of ground terms in canonical form where the symbol that was supposed to disappears has not gone away.

Before giving the general definition, we need to make more precise the notion of subsignature.

Definition 38 *An order-sorted signature $\Omega = ((S', <'), G)$ is called a* subsignature *of an order-soted signature $\Sigma = ((S, <), F)$, denoted $\Omega \subseteq \Sigma$, iff:*

1. *$S' \subseteq S$ and $<' \subseteq <$, and*
2. *for each $(w', s') \in List(S') \times S'$ there is a subset inclusion $G_{w',s'} \subseteq F_{w',s'}$, which we abbreviate with the notation $G \subseteq F$.*

For example, the process of adding variables to a signature gives rise to a subsignature inclusion $\Sigma \subseteq \Sigma(X)$. Likewise, the top completion (resp. kind completion) of a signature Σ gives rise to a signature inclusion $\Sigma \subseteq \Sigma^\top$ (resp. $\Sigma \subseteq \widehat{\Sigma}$).

Here is now our sough-after notion of sufficient completeness, in as general a form as possible.

Definition 39 *Let (Σ, B, R) be a rewrite theory that is weakly ground terminating, and let $\Omega \subseteq \Sigma$ be a subsignature inclusion where Ω has the same poset of sorts as Σ, that is, $\Sigma = ((S, <), F)$, $\Omega = ((S, <), G)$, and $G \subseteq F$. We say that the rules R 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 $t \to^!_{R/B} t'$.*

If Σ is kind-complete, then the above requirement that for each $t \in T_{\Sigma,s}$ there is a $t' \in T_{\Omega,s}$ such that $t \to^!_{R/B} t'$ should apply only to the sorts $s \in [s]$ in each connected component, but not to the kinds $[s]$. Therefore, the sufficient completeness requirement for R modulo B should be placed on a signature Σ *before* kind-completing it to $\widehat{\Sigma}$. This is because, since terms that have a kind $[s]$ but not a sort s, correspond to undefined or error expressions, such as $7/0$, or $p(0)$ for p the predecessor function on natural numbers, it is perfectly possible that a completely well-defined function on the *right sorts* cannot be simplified away when applied to the *wrong* arguments. For example, we can define the predecessor function in a theory with sorts *Nat* and *NzNat* with subsort inclusion $Nat < NzNat$, subsignature Ω of constructors: $s : Nat \longrightarrow NzNat$, $0 : nil \longrightarrow Nat$, and defined function symbol $p : NzNat \longrightarrow Nat$, with the single equation $p(s(x)) = x$, with $x : Nat$. This theory is obviously sufficiently complete, that is, the canonical forms of all ground terms are all Ω-terms. However, in its kind completion we have the term $p(0) : [Nat]$, which is in canonical form, because p is only defined for nonzero natural numbers.

Therefore, if we have identified for our rewrite theory (Σ, B, R) a subsignature of Ω of constructors, a fifth and last requirement should be:

(5) the rules R are sufficiently complete modulo B.

Condition (1) is syntactically checkable, and so is condition (2). Confluence is a decidable property for (Σ, B, R) 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., [2, 5.1]), because of the halting problem

termination is undecidable. Sufficient completeness is also decidable for (Σ, B, R) when R 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 $t \to t' \in R$, any variable x of t occurs only once in t (the so-called *left-linearity* of $t \to t'$).

Maude (see [10]) automatically checks condition (1), can automatically check confluence and condition (2) with its Church-Rosser Checker (CRC) tool [15], 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) [13]. When the equations E are conditional (see §14), 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.

Chapter 14

Conditional Rewriting and Conditional Equational Logic

Unconditional equational logic can express most algebraic properties, but not all of them. Consider, for example, the unsorted algebra of lists of natural numbers, with set of elements $List(\mathbb{N})$, and with a constant *nil* and a binary operation $\_\cdot\_$ interpreted, respectively, as the empty list $\emptyset$ and as list concatenation. Now consider any three lists $l, l', l'' \in List(\mathbb{N})$, and suppose that the list equality $l \cdot l' = l \cdot l''$ holds; then we must also have the equality $l' = l''$. This is called the *left cancellation* property of lists, since in the equality $l \cdot l' = l \cdot l''$ we can "cancel" l on the left. Likewise, if the equality $l' \cdot l = l'' \cdot l$ holds, then we must also have the equality $l' = l''$, which is called the *right cancellation* property.

These two properties cannot be expressed by ordinary (unconditional) equations. They are instead *conditional* properties of the form: (i) if $l \cdot l' = l \cdot l''$, then $l' = l''$, expressing left cancellation; and (ii) if $l' \cdot l = l'' \cdot l$, then $l' = l''$, expressing right cancellation. Such implications, where in general the condition may not be a single equation but a *conjunction* of such equations, are called *conditional equations* and are written

$$(\flat) \quad t = t' \ \ \textit{if} \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n$$

where when $n = 0$ (no equations in the condition), we get the unconditional equation $t = t'$ as a special case. That is, conditional equations *generalize* unconditional ones and provide a strictly more expressive logic. For example, the above two cancellation laws can be expressed by the conditional equations

$$l' = l'' \textit{if} \ \ l \cdot l' = l \cdot l''$$

$$l' = l'' \textit{if} \ \ l' \cdot l = l'' \cdot l.$$

Here is the general definition.

Definition 40 *(Conditional Equations and Conditional Equational Theories). Given a sensible order-sorted signature* Σ*, a* Σ-conditional equation *is an implication formula* $u_1 = v_1 \wedge \ldots \wedge u_n = v_n \Rightarrow t = t'$*, hereafter written in the above form* $(\flat)$*, where* $t = t'$ *and* $u_1 = v_1, \ldots, u_n = v_n$ *are* Σ*-equations. A* conditional equational theory *is then a pair* (Σ, E)*, with* Σ *a sensible order-sorted signature, and* E *a set of conditional* Σ*-equations.*

In a conditional equational theory (Σ, E) all conditional equations $t = t' \ \ \textit{if} \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n$ are implicitly assumed to be *universally quantified* as

$$(\forall x_1 : s_1, \ldots, x_n : s_n) \ t = t' \ \ \textit{if} \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n$$

with $vars(t = t' \ \ \textit{if} \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n) = \{x_1 : s_1, \ldots, x_n : s_n\}$, where, by definition, $vars(t = t' \ \ \textit{if} \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n) = vars(t) \cup vars(t') \cup vars(u_1) \cup vars(v_1) \cup \ldots \cup vars(u_n) \cup vars(v_n)$.

The notions developed for (unconditional) term rewriting and equational logic in Chapter 13 generalize to the conditional case. The goal of this chapter is to explain you how. Let us begin with conditional term rewriting.

14.1 Conditional Term Rewriting and Conditional Equality

Definition 41 *(Conditional Rewrite Rules and Conditional Term Rewriting Systems). Given a sensible order-sorted signature* Σ*, a* Σ-conditional rewrite rule *is an implication formula of the form* $u_1 \rightarrow^* v_1 \wedge \ldots \wedge u_n \rightarrow^* v_n \Rightarrow t \rightarrow t'$*, hereafter written in the form*

$$t \rightarrow t' \ \ \textit{if} \ \ u_1 \rightarrow v_1 \wedge \ldots \wedge u_n \rightarrow v_n$$

where $t \to t'$ is a Σ-rewrite rule, and for each $1 \leq i \leq n$, the terms u_i and v_1 have sorts in the same kind $[s_i]$. Note that in the above notation, each condition $u_i \to^* v_i$ is abbreviated *to just $u_i \to v_i$. A* conditional term rewriting system *(CTRS) is then a pair (Σ, R), with Σ an sensible order-sorted signature, and R a set of conditional Σ-rewrite rules.*

The intuitive idea about how a conditional rewrite rule $t \to t' \ \ if \ \ u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n$ is used is that we can apply it to perform a rewrite $u \to v$ if u and v can be decomposed as $u = C[t\theta]$ and $v = C[t'\theta]$ for a substitution $\theta : vars(t \to t' \ \ if \ \ u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n) \longrightarrow T_{\Sigma(Y)}$, *and* we can prove that *the rule's condition is satisfied* for θ, that is, that the conjunction

$$u_1\theta \to^* v_1\theta \wedge \ldots \wedge u_n\theta \to^* v_n\theta$$

holds. Note that when $n = 0$ (no equations in the condition) this specializes to (unconditional) term rewriting.

Let us see an example. Consider the CTRS (Σ, R), with Σ having sorts *Nat*, *List*, and *Bool*, constants 0 of sort *Nat*, *nil* of sort *List*, and *true*,*false* of sort *Bool*, with unary function symbols s of sort *Nat*, and $card : List \to Nat$, and binary function symbols $\_ \vee \_ : Bool\ Bool \to Bool$, $\_;\_ : Nat\ List \to List$, $\_ \equiv \_ : Nat\ Nat \to Bool$, and $\_ \in \_ : Nat\ List \to Bool$; and with R consisting of the following rules (where $b : Bool$, $n, m : Nat$, and $l : List$):

- $true \vee b \to true$
- $false \vee b \to b$
- $0 \equiv 0 \to true$
- $0 \equiv s(n) \to false$
- $s(n) \equiv 0 \to false$
- $s(n) \equiv s(m) \to n \equiv m$
- $n \in nil \to false$
- $n \in (m; l) \to (n \equiv m) \ \vee \ n \in l$
- $card(nil) \to 0$
- $card(n; l) \to s(card(l)) \ \ if \ \ n \in l \to false$
- $card(n; l) \to card(l) \ \ if \ \ n \in l \to true$

Intuitively, $\_ \vee \_$ is Boolean disjunction, $\_;\_$ is the obvious list constructor, $\_ \equiv \_$ is the equality predicate on natural numbers, $\_ \in \_$ is list membership, and *card* counts the number of *non-repeated elements* in a list. Therefore, if the rules capture this intuition, we should be able to compute, for example, $card(s(0); (0; (s(0); nil))) \to_R^* s(s(0))$.

We can try to apply to the term $card(s(0); (0; (s(0); nil)))$ the next-to-last rule; but we cannot, because when we try to evaluate its condition we get: $s(0) \in (0; (s(0); nil)) \to_R (s(0) \equiv 0) \vee (s(0) \in (s(0); nil)) \to_R false \vee (s(0) \in (s(0); nil)) \to_R (s(0) \in (s(0); nil)) \to_R (s(0) \equiv s(0)) \vee s(0) \in nil \to_R (0 \equiv 0) \vee s(0) \in nil \to_R true \vee s(0) \in nil \to_R true$. But this means that we can apply the last rule to perform the rewrite $card(s(0); (0; (s(0); nil))) \to_R card(0; (s(0); nil))$, and now, since we can easily check that $0 \in (s(0); nil) \to_R^* false$, we can apply the next-to-last rule to perform a second rewrite $card(0; (s(0); nil)) \to_R s(card(s(0); nil))$. In a similar way, the next-to-last rule can be applied to $s(card(s(0); nil))$ to get a third rewrite $s(card(s(0); nil)) \to_R s(s(card(nil)))$, which rewrites in a fourth and last step to $s(s(0))$, as expected.

14.1.1 Conditional Term Rewriting and Conditional Equality as Inference

The interesting question now is: how can we *define* the conditional rewrite relation $\to_R$? The question is interesting because $\to_R$ is *recursive*; that is, to just perform a single rewrite step we need to use its reflexive-transitive closure $\to_R^*$ to evaluate the rule's condition. So $\to_R$ needs somehow to be defined in terms of itself!

This suggests viewing a CTRS (Σ, R) as a theory in a simple logic with two kinds of formulas, namely, formulas of the form $t \to t'$, and of the form $t \to^* t'$, where the *provable formulas* are defined *inductively* by the following three *rules of inference:*

Definition 42 *(Conditional Term Rewriting as Inference). Let (Σ, R) be a CTRS with $\Sigma = ((S, <), F)$ a sensible and kind-complete signature with nonempty sorts, and let $Y = \{Y_s\}_{s \in S}$ be an S-indexed set of variables. Then we say that the sequent $t \to t'$ (resp. $t \to^* t'$) is* provable *in (Σ, R), denoted $(\Sigma, R) \vdash t \to t'$, or, more briefly, $t \to_R t'$ (resp. $(\Sigma, R) \vdash t \to^* t'$, or $t \to_R^* t'$) iff it can be derived by a finite application of the following inference rules:*

- **Reflexivity**. *For each $t \in \bigcup T_\Sigma(Y)$,* $\dfrac{}{t \to^* t}$
- **Replacement**. *For $t \to t' \ \ if \ \ u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n$ a rule in R with set of variables X, $\theta : X \longrightarrow T_\Sigma(Y)$ a substitution, and term decompositions $u = C[t\theta]$ and $v = C[t'\theta]$,*

$$\frac{u_1\theta \to^* v_1\theta \ \ \ldots \ \ u_n\theta \to^* v_n\theta}{C[t\theta] \to C[t'\theta]}$$

- **Transitivity**

$$\frac{t_1 \to t_2 \qquad t_2 \to^* t_3}{t_1 \to^* t_3}$$

We saw in §13.1.2 that equality in an unconditional equational theory (Σ, E) can be trivially reduced to rewriting in the unconditional TRS $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E})$. Can a similar rabbit be pulled out of the hat in the conditional case? The answer is *yes!* But we need to make clear what we mean by $\overrightarrow{E}$ and $\overleftarrow{E}$. Given a conditional equational theory (Σ, E) we define:

- $\overrightarrow{E} = \{t \to t' \ \ if \ \ u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n \mid (t = t' \ \ if \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n) \in R\}$, and
- $\overleftarrow{E} = \{t' \to t \ \ if \ \ u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n \mid (t = t' \ \ if \ \ u_1 = v_1 \wedge \ldots \wedge u_n = v_n) \in R\}$.

Note that we have chosen a left-to-right orientation for the conditions in both cases.

We can now trivially reduce equality in a conditional equational theory to conditional rewriting in the CTRS $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E})$ as follows.

Definition 43 *(The Conditional Equality Relation). Let (Σ, E) be a conditional equational theory with Σ a sensible, kind-complete order-sorted signature with nonempty sorts, and Y an S-indexed set of variables. Then a conditional* E-equality step *is, by definition, a conditional $(\overrightarrow{E} \cup \overleftarrow{E})$-rewrite step $u \to_{(\overrightarrow{E} \cup \overleftarrow{E})} v$, denoted $u \leftrightarrow_E v$, where $u, v \in \bigcup T_{\Sigma(Y)}$.*

Similarly, the conditional E-equality relation*, also called the relation of* equality modulo E*, is, by definition, the binary relation $\to^*_{(\overrightarrow{E} \cup \overleftarrow{E})}$ on $\bigcup T_{\Sigma(Y)}$, denoted $\leftrightarrow^*_E$ or, more briefly, $=_E$. That is, by definition,*

$$u =_E v \quad \Leftrightarrow \quad (\Sigma, \overrightarrow{E} \cup \overleftarrow{E}) \vdash u \to^* v.$$

Notational Convention. Since in many applications a set of conditional equations E will always be used from left to right, we shall often abbreviate the conditional rewrite relations $\to_{\overrightarrow{E}}$ and $\to^*_{\overrightarrow{E}}$, associated to the CTRS $(\Sigma, \overrightarrow{E})$ by, respectively, $\to_E$ and $\to^*_E$. Likewise, we will abbreviate $\to_{\overleftarrow{E}}$, and $\to^*_{\overleftarrow{E}}$, associated to the CTRS $(\Sigma, \overleftarrow{E})$ by, respectively, $\leftarrow_E$ and $\leftarrow^*_E$.

14.1.2 Proofs for Conditional Term Rewriting and Conditional Equality

Recall that in Definition 31 an (unconditional) (Σ, R)-*rewrite proof* was defined as either a term $t \in \bigcup T_{\Sigma(Y)}$, witnessing a 0-step rewrite $t \to^*_R t$, or a sequence of R-rewrite steps of the form $t_0 \to_R t_1 \to_R t_2 \ldots t_{n-1} \to_R t_n$, with $n \geq 1$, witnessing $t_0 \to^+_R t_n$. Likewise, in Definition 32 an (unconditional) (Σ, E)-equality proof was defined as either a term $t \in \bigcup T_{\Sigma(Y)}$, witnessing a 0-step E-equality $t \leftrightarrow^*_E t$, or a sequence of E-equality steps of the form $t_0 \leftrightarrow_E t_1 \leftrightarrow_E t_2 \ldots t_{n-1} \leftrightarrow_E t_n$, with $n \geq 1$, witnessing $t_0 \leftrightarrow^+_E t_n$. But this idea of a proof as a linear sequence of steps will *not* work for either conditional rewriting proofs or conditional equality proofs. Why not? Well, since conditional equality proofs are a special case of conditional rewrite proofs we can focus on why the proof sequence idea breaks down for conditional rewrite proofs.

The problem is that to prove a *single* conditional rewrite step $u \to_R v$ applying a rule, say, $t \to t' \ \ if \ \ u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n$ in R with a substitution θ we also need to *prove* that the substitution instance of the rule's conditions $u_1\theta \to^* v_1\theta \wedge \ldots \wedge u_n\theta \to^* v_n\theta$, expanded out in their reflexive-transitive sense, can be rewritten. But carrying out any of the (possibly many-step) rewrites $u_i\theta \to^* v_i\theta$ may require applying other conditional rules whose conditions also have to be proved. So it is turtles all the way down!

So, what can we do? What we can do is to apply a very useful methodological principle in mathematics that I call the *generalize and conquer principle*. That is, solving a possibly much more general problem than the original one is often *simpler* than solving the original problem you had. So, what is the much more general problem? Obviously, the much more general problem is finding an answer to the question: *what is a proof in any logic*? But to solve this much more general problem we need to solve an even more basic problem, namely, answering the question: *what is a logic*? We may get a little dizzy for a moment with so much generality; but we should not lose our nerve. If the generalize and conquer priciple is worth its salt, the more general the question, the easier should the answer be in many cases.[1]

[1] Obviously *not* in all cases, since there are many generalizations of specific results which either are *false* or no answer as to their truth of falsity is currently known, so that legions of mathematicians are busy trying to give answers. A more general result is a more powerful and valuable result and there is no general free lunch: one often needs to work very hard to achieve that extra generality. The generalize and conquer priciple is obviously not a *general law*. It is instead a very useful *heuristic principle* that can save a lot of effort if one can solve in one blow a possibly infinite number special cases. So the general idea is: you try *first* to think of a more general problem (the more general the better) and try to solve that. If you succeed you are lucky, since you have then solved many problems, including your original one, in one go. But if your conjecture is false or you cannot find a way to solve it at such a general level, then you *work your way down into gradually less general versions of it* containing your original conjecture as a specail case.

So, what is a logic? The word "logic," even in the specific sense of a formal logic, has several senses. There are two sides to it. One the one hand, by a logic we mean a *language of formulas* together with some *rules of inference* that allow us to prove the *theorems* of the logic. On the other hand a logic is a language to reason correctly about *something*, some domain of knowledge. Consider, for example, equational logic. For unconditional order-sorted equational logic, Definition 32 gives us exactly what we need for the first side of this logic, since given an unconditional equational theory (Σ, E), the *theorems* of (Σ, E) are exactly the equations $t = t'$ for wich we can *prove* that $t =_E t'$ by rewriting with the TRS $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E})$. But what is the domain of knowledge about which we can reason correctly with equational logic? Obviously, Universal Algebra! That is, an equational theory (Σ, E) describes syntactically a *semantic domain*, namely, the class of all Σ-algebras that satisfy the equations E. For example, if (Σ, E) is the theory of monoids, then the semantic domain it describes is obviously the class of all monoids.

Building on prior work of Goguen and Burstall, who axiomatized the semantic domain part of a logic in their theory of *institutions* [19], I have given a general, axiomatic answer to the question "what is a logic?" in both its deduction and its semantic domain aspects in the theory of *general logics* that I initiated in [34]. But here we just need to get a handle on the deduction side of a logic, which is quite easy to do in a summarized form,[2] answering in one blow our two questions: (i) what is a proof? and (ii) what is a logic (in the deduction sense)?

A logic $\mathcal{L}$ has a (parametric) syntax in which we can define its *formulas*. For example, the syntax of unconditional order-sorted equational logic is parameterized by the chosen signature Σ, and then its corresponding formulas are the Σ-equations. A *theory* $\mathcal{S}$ in $\mathcal{L}$ is then a choice of syntax together with a set of formulas in such a syntax. For example, a theory (Σ, E) in unconditional equational logic is exactly this: we fix Σ and specify a set E of Σ-equations as the theory's axioms. *Proofs* in $\mathcal{L}$ are then defined by *proof trees*, which can be built up by finite application of the logic's *inference rules*, which are parameterized by each *theory* $\mathcal{S}$.

Definition 44 *The set of (finite)* proof trees *for a theory $\mathcal{S}$ in a logic $\mathcal{L}$ and the head of a proof tree are defined inductively as follows. A* proof tree *is*

- *either an* open goal*, simply denoted as φ, where φ is a formula in the language of $\mathcal{S}$; then, we define $head(\varphi) = \varphi$.*
- *or a* non-atomic *tree with φ as its head, denoted as*

$$\frac{T_1 \quad \cdots \quad T_n}{\varphi} \quad (\Delta)$$

where φ is a formula in the language of $\mathcal{S}$, Δ is an inference rule in $\mathcal{L}$, and $T_1, \ldots, T_n$ are proof trees such that

$$\frac{head(T_1) \quad \cdots \quad head(T_n)}{\varphi}$$

is an instance of Δ for the theory $\mathcal{S}$.

We say that a proof tree is closed *whenever it is finite and contains no open goals.*[3] *Finally, we say that a formula φ is* provable *in $\mathcal{S}$, denoted $\mathcal{S} \vdash \varphi$ if an only if we can build a closed proof tree T such that $head(T) = \varphi$.*

That is, a proof tree with open goals is a *proof attempt*, which sometimes may never be completed, whereas a closed proof tree is an *actual proof*, proving a *theorem* in the theory $\mathcal{S}$.

Notice the difference between φ, an open goal, and $\overline{\varphi}$, a goal closed by a rule without premises. For example, in the inference system below, an equation $t = t$ is still an open goal, but it can be closed as the closed proof tree $\overline{t = t}$ by applying the **Reflexivity** inference rule. Likewise, if $(t = t') \in E$, and θ is a

[2] What here I call a *logic* is a particular style of presenting an *entailment system* in the sense of [34]. For termination purposes, a topic I will consider in §14.3.2, this notion of logic has been used to give a general notion of *operational termination* of a theory in [12].

[3] Open goals appear at the leaves of a proof tree; but they can be *closed* by the application of inference rules with no premisses. For example, an open goal $t = t$ can be closed by applying a Reflexivity inference rule in the uncondtional ordered-sorted equational logic inference system presented below.

substitution applied to $vars(t = t')$, we can then close any open goal of the form $C[t\theta] = C[t'\theta]$ as the closed proof tree $\dfrac{}{C[t\theta] = C[t'\theta]}$ by applying the **Replacement** inference rule.

To bring all these notions home, let us begin by asking the question: what is a possible choice of inference rules Δ for unconditional order-sorted equational logic? A simple choice for Δ is the following, parameterized by a theory (Σ, E) with Σ sensible, kind-complete, and with nonempty sorts:

- **Reflexivity**. For each $t \in \bigcup T_\Sigma(Y)$, $\dfrac{}{t = t}$
- **Replacement**. For either $(t = t') \in E$ or $(t' = t) \in E$, with $X = vars(t = t')$, $\theta : X \longrightarrow T_\Sigma(Y)$ a substitution, and term decompositions $u = C[t\theta]$ and $v = C[t'\theta]$,

$$\frac{}{C[t\theta] = C[t'\theta]}$$

- **Transitivity**. For $t, t', t'' \in \bigcup T_\Sigma(Y)$,

$$\frac{t = t' \qquad t' = t''}{t = t''}$$

Exercise 111 *(Sequence Proofs Generalized as Closed Proof Trees). Given an (unconditional) order-sorted equational theory (Σ, E), show that $u =_E v$, that is, $u = v$ has a proof in (Σ, E) in the sequence sense of Definition 32 iff we can build a closed proof tree T such that $head(T) = u = v$.*

For the theory of semigroups, with unsorted signature Σ having a single binary function symbol $\_ \cdot \_$, and E the associativity axiom $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, build a closed proof tree proving the equality $(x_1 \cdot x_2) \cdot (x_3 \cdot x_4) = (x_1 \cdot (x_2 \cdot x_3)) \cdot x_4$.

So, now that we have answered the more general question of what is a proof in a logic, we have thus found an answer to our original question: what is a conditional rewriting proof? (which answers as a special case the question: what is a conditional equality proof?). The answer is now obvious. Conditional rewriting is a logic where syntax is parameterized by order-sorted signatures Σ, the atomic sentences are of the form $t \to t'$ and $t \to^* t'$, with $t, t' \in T_{\Sigma(Y)}$ having sorts in the same kind, and where we also allow implications of the form $u_1 \to^* v_1 \wedge \ldots \wedge u_n \to^* v_n \;\Rightarrow\; t \to t'$, which we write in abbreviated form as $t \to t' \;\; if \;\; u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n$. A theory in this logic is exactly a CTRS (Σ, R). And the rules of inference to prove those atomic formulas which are the theorems of (Σ, R) are precisely the rules of **Reflexivity**, **Replacement**, and **Transitivity** in Definition 42.

For example, in the CTRS defining the cardinality of lists of natural numbers after Definition 41, the conditional rewrite $s(card(s(0); nil)) \to^* s(s(0))$ has the following (closed) proof tree:

$$\dfrac{\dfrac{\dfrac{}{s(0) \in nil \to false}}{s(card(s(0); nil)) \to s(s(card(nil)))} \qquad \dfrac{\dfrac{}{s(s(card(nil))) \to s(s(0))} \qquad \dfrac{}{s(s(0)) \to^* s(s(0))}}{s(s(card(nil))) \to^* s(s(0))}}{s(card(s(0); nil)) \to^* s(s(0))}$$

Exercise 112 *(An Inference System for Order-Sorted Conditional Equational Logic). As explained in Definition 43, conditional equality proofs in an order-sorted conditional equational theory (Σ, E), with Σ a sensible, kind-complete order-sorted signature with nonempty sorts, are just proofs in the CTRS $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E})$. However, order-sorted conditional equational logic is a logic in its own right and therefore deserves an inference system of its own. Prove that $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E}) \vdash u \to^* v$, therefore proving $u =_E v$ in the sense of Definition 43, iff we can prove $(\Sigma, E) \vdash u = v$ using the following inference system:*

- **Reflexivity**. *For each $t \in \bigcup T_\Sigma(Y)$,* $\dfrac{}{t = t}$
- **Replacement**. *For either $(t = t' \;\; if \;\; u_1 = v_1 \wedge \ldots \wedge u_n = v_n) \in E$, or $(t' = t \;\; if \;\; u_1 = v_1 \wedge \ldots \wedge u_n = v_n) \in E$ with set of variables X, $\theta : X \longrightarrow T_\Sigma(Y)$ a substitution, and term decompositions $u = C[t\theta]$ and $v = C[t'\theta]$,*

$$\frac{u_1\theta = v_1\theta \;\; \ldots \;\; u_n\theta = v_n\theta}{C[t\theta] = C[t'\theta]}$$

- **Transitivity**

$$\frac{t_1 = t_2 \qquad t_2 = t_3}{t_1 = t_3}$$

14.2 Conditional Term Rewriting Modulo Axioms

As we did for unconditional rewrite rules in §13.2, we can raise the level of abstraction of conditional rewriting by *building in* an equational theory B. Here are the definitions.

Definition 45 *Let Σ be a sensible order-sorted signature. A* rewrite theory *is a triple (Σ, B, R), with (Σ, B) an unconditional[4] order-sorted equational theory, and R a set of conditional Σ-rewrite rules.*

Conditional rewriting with *R modulo B* is then be formalized as follows.

Definition 46 *(Conditional Rewriting Modulo B). Let (Σ, B, R) be a conditional rewrite theory such that Σ is sensible, kind-complete and has nonempty sorts; and let $Y = \{Y_s\}_{s \in S}$ be an S-indexed set of variables. Then we say that the sequent $t \to t'$ (resp. $t \to^* t'$) is* provable *in (Σ, B, R), denoted $(\Sigma, B, R) \vdash t \to t'$, or, more briefly, $t \to_{R/B} t'$ (resp. $(\Sigma, B, R) \vdash t \to^* t'$, or $t \to^*_{R/B} t'$) iff it can be derived by a finite application of the following inference rules:*

- **Reflexivity**. *For each $t, t' \in \bigcup T_\Sigma(Y)$ such that $t =_B t'$,* $\dfrac{}{t \to^* t'}$
- **Replacement**. *For $t \to t'$ if $u_1 \to v_1 \wedge \ldots \wedge u_n \to v_n$ a rule in R with set of variables X, $\theta : X \longrightarrow T_\Sigma(Y)$ a substitution, and term decompositions $C[t\theta], C[t'\theta]$ such that $u =_B C[t\theta]$ and $v =_B C[t'\theta]$,*
$$\frac{u_1\theta \to^* v_1\theta \ \ \ldots \ \ u_n\theta \to^* v_n\theta}{u \to v}$$
- **Transitivity**
$$\frac{t_1 \to t_2 \qquad t_2 \to^* t_3}{t_1 \to^* t_3}$$

Exercise 113 *(A CTRS (Σ, R) as a special case of a Rewrite Theory $(\Sigma, \varnothing, R)$). Prove that for a CTRS (Σ, R) with Σ sensible, kind-complete and with nonempty sorts, we have the equivalence:*

$$(\Sigma, R) \vdash t \to^* t' \quad \Leftrightarrow \quad (\Sigma, \varnothing, R) \vdash t \to^* t'$$

where the first deduction uses the inference system of Definition 42 and the second deduction uses the inference system of Definition 46. That is, as in the unconditional case, conditional rewriting modulo axioms contains conditional rewriting as the special case *$B = \varnothing$.*

Exercise 114 *(General Transitivity). Notice that the* **Transitivity** *inference rule for conditional rewrite theories is not fully general, since the first premise must be a one-step rewrite. However, this lack of generality is only apparent. Prove the following for (Σ, B, R) a conditional rewrite theory such that Σ is sensible, kind-complete and has nonempty sorts and $t, t', t'' \in \bigcup T_{\Sigma(Y)}$:*

1. $(\Sigma, B, R) \vdash t \to t' \quad \Rightarrow \quad (\Sigma, B, R) \vdash t \to^* t'$.
2. $(\Sigma, B, R) \vdash t \to^* t' \ \wedge \ (\Sigma, B, R) \vdash t' \to^* t'' \quad \Rightarrow \quad (\Sigma, B, R) \vdash t \to^* t''$.
3. $(\Sigma, B, R) \vdash t \to^* t' \ \wedge \ (\Sigma, B, R) \vdash t' \to t'' \quad \Rightarrow \quad (\Sigma, B, R) \vdash t \to^* t''$.
4. $(\Sigma, B, R) \vdash t \to t' \ \wedge \ (\Sigma, B, R) \vdash t' \to t'' \quad \Rightarrow \quad (\Sigma, B, R) \vdash t \to^* t''$.

(Hints: (i) since the proper subtree relation among proof trees is well-founded, you can use it to prove (2) *by well-founded induction; (ii) consider getting (3) and (4) from (1) and (2)).*

Exercise 115 *As a sanity check, prove that if the rules R in (Σ, R) are unconditional, then:*

1. *$(\Sigma, R) \vdash t \to t'$ in the sense of Definition 42 iff $t \to_R t'$ in the sense of Definition 31, and*
2. *$(\Sigma, R) \vdash t \to^* t'$ in the sense of Definition 42 iff $t \to^*_R t'$ in the sense of Definition 31.*

That is, Definition 42 specializes *to Definition 31 when (Σ, R) is a TRS.*
(Hints: (i) (2) *does not automaticaly follow from (1). That is, since in the inference system of Definition 42 the notation $t \to^*_R t'$ is* purely formal*, we do not* know a priori *that $t \to^*_R t'$ in the sense of $(\Sigma, R) \vdash t \to^* t'$ means the reflexive-transitive closure of $t \to_R t'$ in the sense of Definition 31: this has to be* proved*; (ii) use Exercises 113 and 114 in combination).*

[4] In the full generality of rewriting logic decribed in [8], and also in the Maude language [10], the equational theory (Σ, B) can be conditional. However, for our present uses in Universal Algebra it is sufficient to consider unconditional equational theories (Σ, B).

Exercise 116 *As a sanity check, prove that if the equations E in (Σ, E) are unconditional, then:*

1. *$t \leftrightarrow_E t'$ in the sense of Definition 43 iff $t \leftrightarrow_E t'$ in the sense of Definition 32, and*
2. *$t \leftrightarrow_E^* t'$ in the sense of Definition 43 iff $t \leftrightarrow_E^* t'$ in the sense of Definition 32.*

That is, Definition 43 specializes *to Definition 32 when (Σ, E) is unconditional.*
(Hint: use Exercise 115).

Exercise 117 *Prove that, given a conditional equational theory (Σ, E) satisfying the requirements in Definition 43, the conditional E-equality relation $=_E$ is an* equivalence relation *on $\bigcup T_{\Sigma(Y)}$ and on $T_{\Sigma(Y),[s]}$ for each kind $[s]$ in Σ.*
(Hint: use Exercises 113 and 114 in combination).

Exercise 118 *(Deconstructing Conditional Rewriting and Conditional Equality Modulo; generalizes Exercises 102 and 103). At the heavy price of losing the higher level of abstraction gained by reasoning modulo B, we can "deconstruct" a conditional rewrite theory (Σ, B, R) into a* semantically equivalent *term rewriting system $(\Sigma, R \cup \overrightarrow{B} \cup \overleftarrow{B})$. Similarly, we can deconstruct equational reasoning with E modulo B into standard equational reasoning with $E \cup B$. Prove that if Σ is sensible, kind-complete and has nonempty sorts, then, for any two Σ-terms $t, t' \in \bigcup T_{\Sigma(Y)}$ we have :*

1. $(\Sigma, B, R) \vdash t \rightarrow^* t' \quad \Leftrightarrow \quad (\Sigma, R \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow^* t'$.
2. $(\Sigma, B, \overrightarrow{E} \cup \overleftarrow{E}) \vdash t \rightarrow^* t' \quad \Leftrightarrow \quad t =_{E \cup B} t'$.

(Hints: (i) use well-founded induction on the proper closed proof subtree relation to prove (1); (ii) use Exercise 114-(1) to reformulate the equivalence (1) as the equivalence:

$$(\Sigma, B, R) \vdash t \rightarrow^* t' \vee (\Sigma, B, R) \vdash t \rightarrow t' \quad \Leftrightarrow \quad (\Sigma, R \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow^* t' \vee (\Sigma, R \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow t'$$

since this latter equivalence may make it easier to carry through your well-founded induction; (ii) reduce (2) to (1)).

14.3 Executability Conditions for Conditional Rewrite Theories

What executability conditions should we impose on a conditional rewrite theory (Σ, B, R) —extending conditions (1)–(4) in §13.3 for (Σ, B, R) unconditional— so that it can be efficiently implemented? Some extensions to the condional case are straightforward, and others, like those for (1) and (4), are rather subtle, so I postpone discussing them until §14.3.2 and **??**.

Let us begin with the straighforward extensions. First of all, regarding requirements on B such the existence of a B-matching algorithm nothing has changed: the same requirements apply here for the exact same reasons. Next, consider condition (2) on sort-decreasingness. What should the notion of sort-decreasingness be like for a conditional rewrite rule $t \rightarrow t' \ \ if \ \ u_1 \rightarrow v_1 \wedge \ldots \wedge u_n \rightarrow v_n$ in (Σ, B, R)? There are two answers, one easy and another of more theoretical than practical interest. The easy answer is that, by definition, $t \rightarrow t' \ \ if \ \ u_1 \rightarrow v_1 \wedge \ldots \wedge u_n \rightarrow v_n$ is sort-decreasing iff $t \rightarrow t'$ is so. The more theoretical and more exact answer is that $t \rightarrow t' \ \ if \ \ u_1 \rightarrow v_1 \wedge \ldots \wedge u_n \rightarrow v_n$ is sort-decreasing iff for each substitution θ such that $(\Sigma, B, R) \vdash u_i\theta \rightarrow^* v_i\theta$, $1 \leq i \leq n$, we must have for any sort $s \in S$ that $t\theta : s \Rightarrow t'\theta : s$. Of course, sort decreasingness in the first sense is a sufficient condition for sort-decreasingness in the second sense. From now on I will only call a conditional rule "sort decreasing" *in the first sense*, because, although certainly a stronger condition than strictly necessary, it is very easily checkable, whereas checking the second definition is in general undecidable.

14.3.1 Confluence and the Church-Rosser Property in the Conditional Case

An even easier extension is that of condition (3), confluence, since the *same* definition of confluence and ground confluence modulo B in Definition 35 applies to the conditional case, by just replacing the unconditional rewrite theory (Σ, B, R) in Definition 35 by a conditional rewrite teory (Σ, B, R). Likewise, the notion of *joinability modulo B*, $t \downarrow_{R/B} t'$, extends without any changes to the conditional case.

However, even though the generalization of confluence to the conditional case couldn't be more straightforward, the generalization of its key property, namely, the Church-Rosser property, which, being so easy in the unconditional case, was left as Exercise 106, is not so easy to establish in the conditional case. Since it is such a fundamental property, allowing a very efficient reduction of order-sorted equational logic to just ordered-sorted rewriting, I give a full proof below.

Theorem 15 *(Church-Rosser Propery for Confluent Order-Sorted Conditional Rewrite Theories). Let* $(\Sigma, E \cup B)$ *be a conditional order-sorted equational theory with* Σ *sensible, kind-complete and with nonempty sorts and such that* $(\Sigma, B, \overrightarrow{E})$ *is confluent. Then, for any two terms* $t, t' \in \bigcup T_{\Sigma(Y)}$*, we have the equivalence:*

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

Proof. *Since equational reasoning is more powerful than rewriting, the implication* ($\Leftarrow$) *holds not just for* $(\Sigma, B, \overrightarrow{E})$ *confluent, but for* any $(\Sigma, B, \overrightarrow{E})$*, as shown in the following Lemma:*

Lemma 13 *Let* $(\Sigma, E \cup B)$ *be a conditional order-sorted equational theory with* Σ *sensible, kind-complete and with nonempty sorts. Then, for any two terms* $t, t' \in \bigcup T_{\Sigma(Y)}$*,* $t \downarrow_{E/B} t' \Rightarrow t =_{E \cup B} t'$*.*
Proof. $t \downarrow_{E/B} t'$ *iff there is a term* $v \in \bigcup T_{\Sigma(Y)}$ *such that* $t \rightarrow^*_{E/B} v$ *and* $t' \rightarrow^*_{E/B} v$*, which is equivalent to* $(\Sigma, B, \overrightarrow{E}) \vdash t \rightarrow^* v$ *and* $(\Sigma, B, \overleftarrow{E}) \vdash v \rightarrow^* t'$*. But this implies that* $(\Sigma, B, \overrightarrow{E} \cup \overleftarrow{E}) \vdash t \rightarrow^* v$ *and* $(\Sigma, B, \overrightarrow{E} \cup \overleftarrow{E}) \vdash v \rightarrow^* t'$*, which by Exercise 114 and Exercise 118-(2) implies* $t =_{E \cup B} t'$*, as desired.* □

We can prove the implication ($\Rightarrow$) *by well-founded induction on the proper closed proof subtree relation. Indeed, by definition,* $t =_{E \cup B} t'$ *iff* $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E} \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow t'$ *or* $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E} \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow^* t'$*. That is,* $t =_{E \cup B} t'$ *iff we can build a closed proof tree* T *for the goal* $t \rightarrow t'$ *or the goal* $t \rightarrow^* t'$ *in the theory* $(\Sigma, \overrightarrow{E} \cup \overleftarrow{E} \cup \overrightarrow{B} \cup \overleftarrow{B})$*. We now reason by cases on the shape of* T*, using well-founded induction to assume the result for smaller subtrees. If* T *was built using the* **Reflexivity** *inference rule, then* $t = t'$ *and the result follows trivially. If* T *was built using the* **Replacement** *inference rule we can distinguish three cases:*

1. *We used a rule in* $\overrightarrow{B} \cup \overleftarrow{B}$*, and then* $t \downarrow_{E/B} t'$ *follows trivially from the* **Reflexivity** *inference rule in Definition 46.*

2. *We used a rule in* $\overrightarrow{E}$*, which implies* $(\Sigma, B, \overrightarrow{E} \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow t'$*, which by Exercise 114-(1) implies* $(\Sigma, B, \overrightarrow{E} \cup \overrightarrow{B} \cup \overleftarrow{B}) \vdash t \rightarrow^* t'$*, which by Exercise 118-(1) is equivalent to* $t \rightarrow^*_{E/B} t'$*, which by the* **Reflexivity** *inference rule in Definition 46 trivially implies* $t \downarrow_{E/B} t'$*, as desired.*

3. *We used a rule in* $\overleftarrow{E}$*, which by the same reasoning implies* $t' \rightarrow^*_{E/B} t$*, which by the* **Reflexivity** *inference rule in Definition 46 trivially implies* $t \downarrow_{E/B} t'$*, as desired.*

The only remaining case is a closed proof tree T *built by the* **Transitivity** *inference rule. This means that there is a term* $u \in \bigcup T_{\Sigma(Y)}$ *and* smaller *closed proof trees* T_1, T_2 *proving, respectively, the formulas* $t \rightarrow u$ *and* $u \rightarrow^* t'$*. The well-founded induction hypothesis then gives us* $t \downarrow_{E/B} u$ *and* $u \downarrow_{E/B} t'$*, so there are terms* $v, v' \in \bigcup T_{\Sigma(Y)}$ *such that* $t \rightarrow^*_{E/B} v$*,* $u \rightarrow^*_{E/B} v$*,* $u \rightarrow^*_{E/B} v'$*, and* $t' \rightarrow^*_{E/B} v'$*. But since* $(\Sigma, B, \overrightarrow{E})$ *is confluent, there is then a term* $w \in \bigcup T_{\Sigma(Y)}$ *such that* $v \rightarrow^*_{E/B} w$*, and* $v' \rightarrow^*_{E/B} w$*, which by Exercise 114-(2) trivially implies* $t \downarrow_{E/B} t'$*, as desired.* □

As for the unconditional case, sort-decreasigness, although not strictly required for conditional confluence, is in practice very important, since its absence can easily block conditional confluence. Consider, for example, a CTRS (Σ, R) where Σ has sorts A and B with $A < B$, constants a, b of respective sorts A, B, and function symbols $f, g : A \longrightarrow A$, and $f : B \longrightarrow B$; and, for $x : A$, $R = \{a \rightarrow b, g(x) \rightarrow a, f(x) \rightarrow x \ \ if \ \ g(x) \rightarrow x\}$. Note that the lefthand sides of these rules do not have any function symbols in common. However, we have $f(a) \rightarrow_R a \rightarrow_R b$, and $f(a) \rightarrow_R f(b)$, but, since a and $f(b)$ cannot be further rewritten, we do *not* have $b \downarrow_R f(b)$.

14.3.2 Operational Termination

Termination of a conditional rewrite theory (Σ, B, R) is a bit subtle. Naively, one could imagine that the unconditional notion of termination extends to the conditional case in a straightforward way. That is, that it is enough to require that $\rightarrow_{R/B}$ is well-founded. But this is a mistake. Consider, for example, the very simple unsorted CTRS (Σ, R) where Σ has just three constants a, b, c, and with $R = \{a \rightarrow b \ \ if \ \ a \rightarrow c\}$.

It is easy to see that, since its condition can never be fulfilled, nothing can be rewritten with this conditional rewrite rule. That is, the relation $\rightarrow_R$ is the empty set! Therefore, it is trivially well-founded.

However, any reasonable interpreter mechanizing conditional rewriting will *loop* when trying to rewrite the constant a. Why? Because such a mechanization, to be correct, must, implicitly or explicitly, build a *proof tree* which, if the computation terminates, will eventually be *closed*. Using the rule $a \to b \ \ \textit{if} \ \ a \to c$, the interpreter will try to build a proof tree with the goal $a \to b$ at its root. It will then try to build a proof tree for its condition $a \to c$, but since the only rule is $a \to b \ \ \textit{if} \ \ a \to c$, and $b \neq c$, it can only do so by using the **Transitivity** inference rule to build first a proof tree for $a \to b$, and then another for $b \to c$. But then it is turtles all the way up, since what the interpreter ends up building is the infinite proof tree:

$$\dfrac{\dfrac{\dfrac{\vdots}{a \to b} \qquad b \to^* c}{a \to^* c}}{a \to b}$$

Of course, well foundedness of $\to_R$ is always a *necessary condition*. But, as the above example shows, it is *not* a sufficient condition for a conditional rewrite theory (Σ, B, R) to terminate. What the example actually shows is that —unlike unconditional termination, which might be called *one-dimensional*, since it is the absence of infinite *sequences*— conditional termination is instead what might be called a *two-dimensional* notion, since it is an absence of *infinite proof trees*. That is, we can loop either *horizontally*, as in the unconditional case, by building an infinite sequence of rewrites; or we can loop *vertically*, even just when attempting a single rewrite step. So we need a better notion of termination. A number of conditional temination notions have been proposed (see, e.g., [41] for a very good survey up to 2002); but some of them are clearly wrong. For example, there is one called "effective termination" according to which the above example terminates!

So, what can we do? Why, of course, apply the generalize and conquer principle! Why not trying to find an adequate notion of termination, not just for conditional rewriting, but for any logic? It may in the end be simpler; and it will certainly be much more widely applicable.

The notion of termination we are looking for should capture the idea that a theory $\mathcal{S}$ in a logic $\mathcal{L}$ *terminates* iff, given any goal formula φ in the language of $\mathcal{S}$, a machine implementation of the logic, an *interpreter* for $\mathcal{L}$, will never loop when trying to find a closed proof T proving goal φ. Durán, Lucas, Marché, Urbain, and I defined this notion in [12] and called it *operational termination*, to emphasize the fact that it captures the practical notion of a machine implementation of $\mathcal{L}$ never looping when trying to prove a goal φ in the language of $\mathcal{S}$.

More concretely, we also assumed that all the inference rules Δ of $\mathcal{L}$ are such that it makes sense for a machine implementation to try to solve the inference rule's conditions for an instance of any such rule Δ of the form

$$\dfrac{\varphi_1 \quad \cdots \quad \varphi_n}{\varphi}$$

by working *sequentially from left to right* by trying to find first a closed proof tree T_1 for subgoal φ_1, then another such T_2 for subgoal φ_2, and so on, until (if successful) finding a closed proof tree T_n for subgoal φ_n, thus obtaining a closed proof tree for goal φ. We called the proof attempts that the interpeter can buidl this way *well-formed proof trees*. Lack of operational termination then coincides with the presence of an *infinite* well-formed proof tree, such as the one shown above for the goal $a \to b$ using the conditional rule $a \to b \ \ \textit{if} \ \ a \to c$. Here are the precise definitions:

Definition 47 *A proof tree T is a* proper prefix *of a proof tree T' if there are one or more open goals $\varphi_1, \ldots, \varphi_n$ in T such that T' is obtained from T by replacing each φ_i by a non-atomic proof tree T_i having φ_i as its head. We denote this as $T \subset T'$.*

An infinite proof tree *is an infinite increasing chain of finite trees, that is, a sequence $\{T_i\}_{i \in \mathbb{N}}$ such that for all i, $T_i \subset T_{i+1}$.*

Definition 48 *We say that a proof tree T is* well-formed *if it is either an open goal, or a closed proof tree, or a proof tree of the form*

$$\dfrac{T_1 \quad \cdots \quad T_n}{\varphi} \quad (\Delta)$$

where for each j, $1 \leq j \leq n$, T_j is itself well-formed, and there is an $i \leq n$ such that T_i is not closed, for any $j < i$ T_j is closed, and each of the $T_{i+1}, \ldots, T_n$ is an open goal. An infinite proof tree is well-formed *if*

it is an ascending chain of well-formed finite proof trees. $\mathcal{S}$ is called operationally terminating *if no infinite well-formed tree for $\mathcal{S}$ exists.*

Our application of the generalize and conquer principle thus solves in one blow the problem of finding a proper notion of termination, not just for a CTRS (Σ, R), or, more generally, for a conditional rewrite theory (Σ, B, R), but for theories in many other logics. For example, the original stimulus in [12] came from the need to find a proper notion of termination for theories in *membership equational logic* [36, 7], which generalizes order-sorted equational logic and combines equalities $t = t'$ and membership predicates $t : s$ for s a sort. For another example, if our logic $\mathcal{L}$ is Horn logic, then a (pure) Prolog interpreter is a Horn logic interpreter which evaluates subgoals from left to right. Operational nontermination of a Horn theory $\mathcal{S}$, whose sentences are clauses of the form $P(\vec{t}) \;\; if \;\; P_1(\vec{t_1}) \wedge \ldots \wedge P_n(\vec{t_n})$, is then the possibility that a Horn logic implemention (not necessarily a given Prolog interpreter) *could* generate an infinite well-formed proof tree when trying to solve a some goal formula φ in the language of $\mathcal{S}$.

Note that for a goal φ in a theory $\mathcal{S}$ of logic $\mathcal{L}$ there could simultaneously exist a closed proof tree T proving φ, and an infinite well-formed proof tree T^∞ attempting to prove the same goal φ. This is exactly what happens, even for unconditional rewrite theories (Σ, B, R), when (Σ, B, R) is weakly teminating: there may exist a terminating sequence $t \rightarrow_{R/B}! t'$ and also an infinite rewrite sequence from t trying to reach t'. Likewise, this can happen for conditional rewrite theories, Hon logic, theories, and theories in many logics.

Therefore, whether an interpreter happens to loop trying to build an infinite proof tree may crucially depend on the interpreter's *strategy*, which dictates the choices the interpreter makes. Since — because of their depth first search strategy— Prolog implementations are *incomplete* (that is, they may fail to find a proof for a goal φ when such a proof exists), a closed proof tree T for a given goal φ may exist, but a Prolog interpreter may "go down the deep end" building a T^∞ and never finding the existing closed proof tree T. Of course, if the Prolog program $\mathcal{S}$ is operationally terminating this will never happen, so that the Prolog interpreter will then, for any goal φ in the language of $\mathcal{S}$, either: (i) stop with a closed proof tree T proving goal φ, that is, proving $\mathcal{S} \vdash \varphi$, or (ii) fail to find any such closed proof tree *in finite time*; that is, after a finite number of failed proof attempts the interpreter will stop with *failure*, showing that $\mathcal{S} \nvdash \varphi$.

Exercise 119 *(Unconditional Termination* is *Operational Termination). Prove than an unconditional rewrite theory (Σ, B, R) is teminating in the standard well-founded sense of Definition 36 iff, when viewed as a special case of a conditional rewrite theory, it is operationally terminating using the inference system of Definition 46.*

Bibliography

[1] J. Avigad and S. Feferman. Gödel's functional ("Dialectica") interpretation. In S. Buss, editor, *Handdbook of Proof Theory*, pages 337–405. Elsevier, 1998.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[3] J. Barwise. An introduction to first-order logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 5–46. North-Holland, 1977.

[4] G. Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[5] G. Birkhoff. *Lattice Theory*. American Mathematical Society, 1948.

[6] P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.

[7] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.

[8] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theor. Comput. Sci.*, 360(1-3):386–414, 2006.

[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] R. Dedekind. *The Nature and Meaning of Numbers*. Dover, 1963. Part II of *Essays on the Theory of Numbers* in the Dover edition.

[12] 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.

[13] 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.

[14] F. Durán, S. Lucas, and J. Meseguer. Termination modulo combinations of equational theories. In *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, volume 5749 of *Lecture Notes in Computer Science*, pages 246–262. Springer, 2009.

[15] 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.

[16] F. Durán and J. Meseguer. On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. *J. Algebraic and Logic Programming*, 2012. Available online 11 Jan. 2012.

[17] K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.

[18] 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.

[19] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[20] J. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985. Preliminary version in: *SIGPLAN Notices*, July 1981, Volume 16, Number 7, pages 24-37.

[21] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[22] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer, 2000.

[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] H. Hermes. *Enumerability, Decidability, Computability*. Springer Verlag, 1969.

[27] T. Jech. About the axiom of choice. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 345–370. North-Holland, 1977.

[28] P. Johnstone. *Notes on Logic and Set Theory*. Cambridge University Press, 1987.

[29] F. W. Lawvere. An elementary theory of the category of sets. *Proceedings, National Academy of Sciences*, 52:1506–1511, 1964.

[30] 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.

[31] F. W. Lawvere. Continuously variable sets: Algebraic Geometry = Geometric Logic. In *Proc. Logic Colloquium (Bristol 1973)*, pages 135–153. North-Holland, 1975.

[32] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic*. Springer Verlag, 1992.

[33] J. Meseguer. Order completion monads. *Algebra Universalis*, 16:63–82, 1983.

[34] J. Meseguer. General logics. In H.-D. E. et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[35] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[36] 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.

[37] J. Meseguer. Twenty years of rewriting logic. *J. Algebraic and Logic Programming*, 81:721–781, 2012.

[38] 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.

[39] J. Meseguer and G. Roşu. A total approach to partial algebraic specification. In *Proc. ICALP'02*, pages 572–584. Springer LNCS 2380, 2002.

[40] P. D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B, Chapter 11*. North-Holland, 1990.

[41] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer Verlag, 2002.

[42] B. Russell. *Introduction to Mathematical Philosophy*. George Allen & Unwin, 1920.

[43] 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.

[44] 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.

[45] B. van der Waerden. *Algebra (vols. I and II)*. Ungar, 1970.

[46] A. van Deursen, J. Heering, and P. Klint. *Language Prototyping: An Algebraic Specification Approach*. World Scientific, 1996.

[47] J. van Heijenoort, editor. *From Frege to Gödel — A Source Book in Mathematical Logic, 1879–1931*. Harvard Univ. Press, 1967.

[48] P. Winkler. *Mathematical Mind-Benders*. A. K. Peters Ltd., 2007.

[49] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.