CATEGORIES, RELATIONS AND DYNAMIC PROGRAMMING

by

Oege de Moor

Technical Monograph PRG-98
ISBN 0-902928-76-7

April 1992

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
England

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
England

Electronic mail: oege@comlab.ox.ac.uk

A thesis submitted for the degree
of Doctor of Philosophy

Oege de Moor
St. John's College
Michaelmas 1991

Categories, Relations and Dynamic Programming

Abstract

Dynamic programming is a strategy for solving optimisation problems. In this thesis, it is shown how many problems that may be solved by dynamic programming are instances of the same abstract specification. This specification is phrased using the calculus of relations offered by topos theory. The main theorem that underlies dynamic programming can then be proved by straightforward equational reasoning. This is the first contribution of this thesis: to provide an elegant formulation of dynamic programming in a categorical setting.

The generic specification of dynamic programming makes use of higher–order operators on relations, akin to the *fold* operators found in functional programming languages. In the present context, a data type is modelled as an initial F–algebra, where F is an endofunctor on the topos under consideration. The mediating arrows from this initial F–algebra to other F–algebras are instances of fold — but only for total functions. Can we generalise to relations? For a *regular* category $\mathcal{E}$, it is possible to construct a category of relations *Rel*$(\mathcal{E})$. When a functor between regular categories is a so–called *relator*, it can be extended (in some canonical way) to a functor between the corresponding categories of relations. Applied to an endofunctor on a topos, this process of extending functors preserves initial algebras, and hence fold can be generalised from functions to relations. This is the second contribution of this thesis: to show how category theory facilitates a smooth generalisation of functional concepts to relations.

It is well–known that the use of dynamic programming is governed by the *principle of optimality*. Roughly, the principle of optimality says that an optimal solution is composed of optimal solutions to subproblems. In a first attempt, we formalise the principle of optimality as a distributivity condition. This distributivity condition is elegant, but difficult to check in practice. The difficulty arises because we consider minimum elements with respect to a preorder, and therefore minimum elements are not unique. Assuming that we are working in a Boolean topos, it can be proved that monotonicity implies distributivity, and this monotonicity condition is easy to verify in practice. This is the third contribution of this thesis: to develop practical results about minimisation in preorders.

0 Preface

Dynamic programming is a problem solving technique that originated iu operations research. Typical applications of dynamic programming are optimisation problems whose specifications can be factorised into three consecutive components. In the first component, one generates a set of combinatorial objects, for example all binary search trees with a given frontier, or all partitions of an integer. In the second component, one assigns a cost to each of these objects. In the third and final component of the specification, one selects an object of minimum cost. The objective of dynamic programming is to merge these three components, to obtaiu a more efficient way of compnting an object of mininnm cost. Informally, this is possible if an optimal solution is composed of optimal solutions to subproblems. This property, which is known as the *principle of optimality*, can often be phrased as a distributivity condition on the operators involved in the specification. Ever since dynamic programming was iuvented in the fifties, researchers have tried to exploit this observation to give a precise characterisation of dynamic programming. This thesis reports on another attempt to achieve such a characterisation. Our starting point is somewhat different from earlier work on dynamic programming, since our results are inspired by modern methods for the derivation of programs from specifications.

0.1 Foundations

0.1.1 The Bird–Meertens Formalism The programming methods that inspired this work stem from the area of functional programming, and in particular from the formalism designed by Bird, Meertens and others [13, 14, 15, 16, 38, 55, 68]. This formalism is a calculus built around initial data types and homomorphisms on those data types. The data types are typically various forms of trees, and the homomorphisms on trees are often called *fold* operators. The advantage of programming in terms of fold is that one may nse initiality to prove eqnality of programs, thus avoiding the tedious steps involved in ordinary inductive proofs.

0.1.2 The Need for Relations In principle, the Bird-Meertens formalism only involves total functions, an assumption which greatly simplifies the mathematical laws used in deriving algorithms from specifications. For the treatment of dynamic programming, however, this restriction is too severe, and we shall need a calculus of arbitrary relations instead of just total functions. Two reasons for generalising from functions to relations are discussed below.

A purely functional style is inadequate because in many optimisation problems one has to deal with nondeterminism. For example, consider the problem of constructing an optimal binary search tree representing a given set. There may well be two different trees of minimum cost that represent the same set, and hence the specification is nondeterministic. The situation cannot be modelled by a function from input to output; rather, it is a relation.

A second reason for considering relations instead of functions is the structure of certain proofs. There are deterministic programming problems (functions) where it is helpful to consider nondeterministic programs (relations) in passing from specification to implementation. A typical instance occurs when considering program inverses [27, 28, 33, 48]: not all functions have an inverse, but all relations do have a converse. In dynamic programming, program inverses are often helpful in defining the set of combinatorial objects.

0.1.3 Sets and Power Transpose A calculus of initial data types and relations does not necessarily involve a form of set theory. In specifying dynamic programming problems, however, it is quite common to specify sets of combinatorial objects by equations of the form

$$gen(y) \;=\; \{\, x \mid f(x) = y \,\}\ .$$

This observation motivates the following requirement for our programming formalism: there exists an isomorphism between relations $A \rightarrow B$ and set–valued functions $A \rightarrow P(B)$. To see how this relates to the above equation for gen , write Λ for the operator that sends a relation to the corresponding set–valued function, and write R° for the converse of R. Then the function gen can be written as

$$gen = \Lambda(f^\circ)\ .$$

The isomorphism Λ, which is called *power transpose*, allows one to give simple equational proofs of identities in set theory.

0.1.4 Minimisation in Preorders The existence of relations and power transpose makes it possible to reason about the first two components of the specification of dynamic programming: generating a set of combinatorial objects, and assigning a cost to each of these objects. However, the third component, which is the selection of an optimal element from a set, cannot be directly described in terms of these primitives. Let $R : A \rightarrow A$ be a preorder (a reflexive and transitive relation), and let $min(R) : P(A) \rightarrow A$ be the relation that maps its argument set to its minimum elements:

$$a(min(R))x \;=\; a \in x \,\wedge\, (\forall b \in x : a\ (R)\ b)\ .$$

more general approaches to program construction. Also, the application of our results seems more mechanical, mostly owing to the algebraic framework.

0.2 Background

The idea to use toposes as a starting point for the investigation of dynamic programming is not immediately obvious. It gradually developed, and since this development seems relevant to the conclusions of this thesis, it will be summarised below. Tbe research that led to this thesis started in the summer of 1988 with the study of various dynamic programming algorithms in Richard Bird's calculus of functions, a subject which he had already touched upon in [15]. This initial exploration [17, 72, 74] identified two obstacles standing in the way to further progress: first, the level of abstraction in the calculus was too low. Many results looked alike, but they could not be expressed as a single theorem. Second, the insistence on total functions hampered the treatment of optimal solutions that are not unique. It seemed expedient, therefore, to change the formalism, and my first attempt was to generalise the basic identities of Bird's calculus from total functions to relations. While this approach solved the problem of non–unique optimal solutions, it failed to cope with the lack of abstraction.

The main result that needed a proof was about the generalisation of fold operators from functions to relations. It was clear that a set theoretical approach was unhelpful here, and many people (in particular Joseph Goguen) suggested the use of category theory. Indeed, with a naive categorical construction of relations, it turned out to be quite easy to prove the desired theorem. Charles Wells encouraged me to show that the assumptions underlying the naive construction reduced to a topos that satisfies tbe axiom of choice. These results were reported in a first draft of this thesis [73].

It then took some time to decide which facts about categorical relations are well–known and which are not. Fortunately, the textbook by Freyd and Šćedrov appeared around this time [40], and most results are in there. Aurelio Carboni kindly gave me access to some unpublished work [24], which showed that splitting the coreflexives in my naive construction of relations yields the regular reflection of a category with finite limits. In all, this study of related work in category theory convinced me that it was better to abandou my naive approach: the most effective proof method is a proof by reference. Also, the standard approach to relations avoids the axiom of choice.

There are, I think, two lessons to be learned from the way in which this research evolved. First, the development of new theoretical concepts in programming methods should be guided by the study of concrete examples. This allows one to select

Note that there may be more than one minimum element a in a set x; one could therefore think of $min(R)$ as a non–deterministic mapping that selects some optimal element from its argument set. To model this relation in onr calculus, we need some form of universal quantification.

0.1.5 Categorical Relations and Toposes What is an appropriate domain of discourse for reasoning about initial data types, relations, power transpose and minimisation? It seems reasonable to start with a category $\mathcal{E}$, where the objects are types, and the arrows are functions. Informally, one could think of $\mathcal{E}$ as an abstract view of a functional calculus like the Bird–Meertens formalism. A data type can then be modelled as an initial F–algebra, where F is an endofunctor on the category under consideration. The mediating arrows from this initial algebra to other F–algebras are instances of fold — but only for total functions. Can we generalise to relations?

For a *regular* category C, it is possible to construct a category of relations $Rel(C)$. Conversely, every category of relations arises in this way. Hence, the assumption that relations exist in our programming formalism means that $\mathcal{E}$ is regular. When a functor between regular categories is a so–called *relator*, it can be extended (in some canonical way) to a functor between the corresponding categories of relations. A regular category where relations and set–valued functions are isomorphic is called a *topos*. When applied to an endofunctor on a topos, the process of extending functors to relations preserves initial algebras, and hence fold can be generalised to relations.

In a topos, one has all sorts of logical operators on relations, in particular universal quantification. It is therefore possible to define the relation $min(R)$, and to reason about its properties. We use this observatiou to formalise the principle of optimality. As mentioned above, the principle of optimality says that an optimal solutiou is composed of optimal solutions to subproblems. In a first attempt, we model the principle of optimality as a distributivity condition. This distributivity condition is elegant, but difficult to check in practice. The difficulty arises because we consider minimum elements with respect to a preorder, and therefore minimum elements are not unique. Assuming that we are working in a *Boolean* topos, it can be proved that monotonicity implies distributivity, and tbis monotonicity condition is easy to verify in practice.

The brief account given above already illustrates bow tbe categorical calculus of relations (as offered by topos theory) meets the requirements of dynamic programming iu an almost perfect manner. This is the main contribution of this tbesis: to recognise that the categorical calculus of relations is an appropriate setting for applications in algorithm design. The result is an attractive treatment of dynamic programming, which extends earlier work of others by making tbe connection with precisely those concepts which are relevant to the applications under consideration. Second, (and this came as a surprise to me) many of the mathematical structures identified in this way are already in existence, although they were invented for completely different purposes. The search for such connections is of course sound scientific practice, but there is also a pragmatic reward: one can present the main results and applications without elaborating the foundations.

The structure of this thesis reflects that viewpoint. The first part presents the results about dynamic programming and their applications. Although the presentation assumes a good deal of category theory, I believe the essence of this work can be understood by anyone who has a nodding acquaintance with categories and functors. Readers who are unfamiliar with more advanced concepts like *regular category* or *topos with a natural numbers object* can just think of the category of sets and total functions.

The second part of this thesis presents the technical details, and it reviews those facts about categorical relations and toposes that are relevant to the present discussion. This part is therefore directed towards computing scientists who (like myself) are not yet skilled in the art of diagram chasing. It is for this reason that many of the proofs are given in much detail, in a calculational style that is familiar to students of formal methods in computing. Sometimes these rigorous calculations have been enlivened by a diagram, both to space out the formulae and as an aid to type checking the proofs.

0.3 Acknowledgements

Three supervisors have guided the research that led to this thesis. Doaitse Swierstra was my *afstudeer* supervisor at Utrecht University, and he stimulated my interest in the formal aspects of program construction. I am grateful to him for the countless discussions we had since 1987, and for his support and friendship. Richard Bird supervised the first two years of my D.Phil. work at Oxford, and I am grateful to him for his tireless efforts to teach me taste and method. From the time I took up an interest in category theory, Tony Hoare has been a constant source of inspiration and sage advice. For the last eight months of this research, he was also formally my supervisor.

I would also like to acknowledge helpful discussions with Aurelio Carboni, Peter Freyd, Paul Gardiner, Joseph Goguen, Clare Martin, Lambert Meertens, Doug Smith and Charles Wells. All have made significant contributions to my understanding of the subject. Four anonymous referees, who commented on earlier parts of this thesis when I submitted them for publication, have made suggestions that led to substantial improvements. In particular, all four insisted on the use of well-

established results from category theory, in preference to a more naive approach that I developed with the applications in mind.

The generous financial assistance of British Petroleum International Ltd., the British Council, the Dutch STOP project, Lincoln College and St. John's College is gratefully acknowledged.

Contents

0 Preface i
0.1 Foundations i
0.2 Background iv
0.3 Acknowledgements v

I Dynamic Programming 1

1 Theory 3
1.1 Introduction 3
1.2 Dynamic Programming 7
1.3 Constructing Selectors from Preorders 13
1.4 Monotonicity implies Distributivity 16
1.5 Summary 18

2 Applications 21
2.1 String–to–String Correction 21
2.2 Loading 24
2.3 Bracketing 26

3 Discussion 32
3.1 Related Work 32
3.2 Future Research 35
3.3 Conclusions 38

II Technical Details 41

4 Introduction 43
4.1 Notation 43
4.2 Overview 44

5 Regular Categories 46
5.1 Subobjects, Images and Covers 47
5.2 Relations over a Regular Category 52
5.3 Extension of Functors 56

5.4 Allegories 63

6 Toposes **68**
6.1 The Definition of a Topos 69
6.2 Cross–operators 75
6.3 Relational Algebras 87
6.4 Power Allegories 91
6.5 Existential Image 98

7 Representing Partial Relations **107**
7.1 Simple–Relation Classifiers 107
7.2 Relation Totalisers 110
7.3 Properties of Relation Totalisers 117

8 Maximisation in Preorders **121**
8.1 Maximum Elements 122
8.2 Selectors 128
8.3 Monotonicity implies Distributivity 137

9 Dynamic Programming **145**
9.1 The Basic Theorem 145
9.2 Application: Text Formatting 149
9.3 A more Practical Theorem 156

10 Discussion **161**
10.1 Which Functors preserve Division? 161
10.2 Does every Topos have a Weak Relation Totaliser? 162
10.3 What are Selectors? 163

Part I

Dynamic Programming

1 Theory

1.1 Introduction

Dynamic programming is a strategy for solving optimisation problems [9. 32, 35]. It is based on the observation that in many optimisation problems an optimal solution is composed of optimal solutions to subproblems. This property has been called the *principle of optimality* by Richard Bellman [9, 91], who invented dynamic programming in the fifties. If the principle of optimality is satisfied, one may compute au optimal solution to the whole problem by decomposing it into subproblems, solving these recursively, and composing the partial solutions into an optimal solution for the whole problem. Dynamic programming is a degenerate case of *divide–and–conquer* in that one considers *all* possible decompositions of the argnment, not just a single one.

Typical applications of dynamic programming include text formatting, molecular sequence comparison, knapsack, and the construction of optimal binary search trees [49]. Given the variety of these applications, one wonders whether it is at all possible to develop a small, coherent mathematical theory to support dynamic programming. Fortunately, the answer is yes. The problems that may be solved by dynamic programming are all instances of a single specification, or slight variations thereof. In this thesis we shall phrase that generic specification using the calculus of relations in a topos. The reasons for nsing topos theory (and not ordinary set theory) are purely pragmatic: toposes provide the right primitive operators, and many of tbe basic facts we shall need have been known to topos theorists for some years.

Rather than presenting a list of technical definitions, we shall introduce the generic specification of dynamic programming by means of an example; the technicalities will come later. Consider the problem of breaking a list of words into lines to form a paragraph [12, 15, 60]. There are many ways of doing this, and we are interested in forming a paragraph with as little wbite space as possible. For expository reasons, we shall first stndy the problem of determining the minimum amount of white space rather than constrncting a paragraph which realises that minimum. Later on we shall see how an optimal paragraph can be constructed.

There are three data types involved in the problem statement: words, lines and paragraphs. For now it is irrelevant how words are represented, and we jnst assume the existence of a set W of words. Lines are sequences of words, and the set of all lines is denoted W^+. In turn, paragraphs are sequences of lines, and the set of all paragraphs is written W^{++}. Both lines and paragraphs are assumed to be non–empty sequences.

Here is the specification for which we intend to derive an algorithm:

$$k(z) \quad = \quad min\ \{\, spacc(x) \mid x \in layouts(z)\,\}\ .$$

The function k takes a line and returns a number which is either natural or infinity. (The need for infinity will hecome clear when we get down to technical matters.) The argument of k, here called z, is the line of words which is to be broken into a paragraph. In the sequel such a paragraph will be called a *layout* of z. The result of k is the minimum amount of white space in a layout of z. How is that result computed? Well, one way is to first generate all possible layouts of z with the function *layouts*. It takes the line z and it returns the set of all possible layouts of z. For each of these layouts (say x) we compute the amount of white space $space(x)$. Finally, take the minimum of the set of numbers generated in this way: the minimum amount of white space in a layout of z.

To define the function *layouts*, we first introduce *concat*, which is familiar to functional programmers [18]. It takes a paragraph, and it concatenates the component lines to form a single line. For example, we have the identity

$$\begin{array}{ll} & concat\ [\ [\text{“this”},\ \text{“is”}\ ], \\ & \qquad\quad\ [\text{“a”}\ ], \\ & \qquad\quad\ [\text{“text”},\ \text{“layout”}\ ]\ ] \\ = & \\ & [\text{“this”},\ \text{“is”},\ \text{“a”},\ \text{“text”},\ \text{“layout”}\ ]\ . \end{array}$$

The set of all layouts of z is precisely the set of those paragraphs x for which $concat(x) = z$. Writing out the definition, it becomes clear that *layouts* is the power transpose of the converse of *concat*:

$$\begin{array}{lllll} layouts(z) & = & \{\,x \mid concat(x) = z\} & = & \{\,x \mid z(concat^\circ)x\,\} \\ & = & \Lambda(concat^\circ)(z)\ . & & \end{array}$$

Here Λ stands for the *power transpose* operator, which takes a relation $A \rightarrow B$ to the corresponding set–valued function $A \rightarrow P(B)$, and $R^\circ : B \rightarrow A$ denotes the *converse* of a relation $R : A \rightarrow B$. From now on, we shall call the power transpose of the converse of R the *inverse image function* of R:

$$Inv(R) \quad = \quad \Lambda(R^\circ)\ .$$

In providing a formal characterisation of the function $concat : W^{++} \to W^{+}$, we start with a precise definition of the data type of paragraphs W^{++}. It is defined as the initial F–algebra, where $F : Set \to Set$ is the functor given by

$$F(A) \;=\; W^{+} + (W^{+} \times A) \;.$$

We shall denote this initial algebra by $\mu(F) : F(T) \to T$. Initial algebras for endofunctors are well–docnmented in the literature [62, 66], so we shall confine ourselves to briefly recalling the definition. For each F–algebra $h : F(A) \to A$ there exists precisely one morphism $\llparenthesis h \rrparenthesis_F$ which makes

$$\begin{tikzcd}
F(T) \arrow[r, "\mu(F)"] \arrow[d, "F\llparenthesis h \rrparenthesis_F"'] & T \arrow[d, "\llparenthesis h \rrparenthesis_F"] \\
F(A) \arrow[r, "h"'] & A
\end{tikzcd}$$

commute. The mediating arrow $\llparenthesis h \rrparenthesis_F$ is pronounced "fold–aitch", and it is said to be a *fold*. The subscript F is not prononnced and it will be omitted if there is no chance of confusion. Folds are interesting [64], because they are similar to the *fold* or *reduce* operators found in fnnctional programming languages like Miranda[1] or Hope [18, 37, 94].

Let us illustrate this with two examples. For the given choice of F, fold could be informally characterised by

$$\begin{aligned}
&\llparenthesis [m, \oplus] \rrparenthesis\; [l_1, l_2, \ldots, l_n] \\
= \;& \\
&l_1 \oplus (l_2 \oplus (\ldots \oplus m(l_n))) \;.
\end{aligned}$$

That is, apply the function m to the last element of the given sequence of lines, and sum the remaining lines from right to left with the binary operator $\oplus$. The function *concat*, which takes a paragraph and concatenates its component lines, can be expressed using fold:

$$concat \;=\; \llparenthesis [1, +\!\!+] \rrparenthesis \;.$$

ft leaves the last line of the paragraph nnchanged (the identity 1), and it concatenates the component lines from right to left, using the binary operator $+\!\!+$ which concatenates two lines to form a single one, e.g.

$$[w_1, w_2] +\!\!+ [w_3, w_4, w_5] \;=\; [w_1, w_2, w_3, w_4, w_5] \;.$$

[1] Miranda is a trademark of Research Software Limited.

As a second example of tbe use of fold, consider the function *space*, which takes a paragraph and returns the amount of white space in that paragraph. It is defined

$$
\begin{array}{rcl}
space & = & (\!|[t,\otimes]|\!) \\
 & & \text{where } \begin{array}{rcl} l \otimes n & = & f(l) + n \\ f(l) & = & (optlen - length(l))^2 \\ t(l) & = & \begin{cases} 0 & \text{if } \; length(l) \leqslant optlen \\ \infty & \text{otherwise} \;. \end{cases} \end{array}
\end{array}
$$

In words, the total amount of white space in a paragraph is the sum of the white space in the component lines, except for the last line, which does not count unless it is too long. The white space in a single line l is returned hy the functiou f. The precise definition of f is unimportant for the present discussion, but one could take, for example, the square of the difference between the optimum line length and the actual length of the given line l.

In summary, the problem of determining the minimum amount of white space in a text layout can be specified as

$$k \;=\; min \circ \exists(space) \circ Inv(concat) \;.$$

The inverse image function $Inv(concat)$ generates all possible layouts of the argument. For each of these layouts, *space* computes the amount of white space. (As is usual in topos theory, the existential image functor $Set \to Set$ is denoted by $\exists$. Functional programmers can think of $\exists$ as the *map* operator on sets.) Finally, the function *min* takes the minimum of the set that has been generated in this way.

One could argue that we now have an executable specification, for it is not difficult to give programs that implement the three main components of the definition of k. However, generating all possible layouts is grossly inefficient. Consider the following recursion equation which describes a more economical approach to tbe computation of k:

$$k(z) \;=\; t(z) \sqcap min\,\{\, f(u) + k(v) \mid u +\!\!\!+\, v = z \,\} \;.$$

The minimum amount of white space in a layout of z is the minimum of two numbers: $t(z)$ and the minimum of a set. Intuitively, this means that either you don't split z or you do split it. If you do not split it, z is the only line in the paragraph, and the amount of white space is just $t(z)$. If you do split it, you consider all possible splittings of z into two sequences, say u and v. (Remember that lines are *non–empty* sequences, so neither u nor v is empty.) The first of these sequences (u) will be the first line of the paragraph; the amount of white space in that first line is $f(u)$. For the remainder v of z you recursively compute $k(v)$. The snm $f(u)+k(v)$ is then the minimum amount of wbite space in a layout of z, given that u is the first line of that layout. Hence,

$$min\ \{\, f(u)+k(v) \mid u \mathbin{+\!\!\!+} v = z \,\}$$

is the minimum amount of white space in a layout of z, given that the layout consists of at least two lines.

If one translates the recursion equation for k directly into a programming language, the resulting program will still he inefficient. The reason is that k gets computed many times over for the same argument. This problem can be alleviated by storing the value of k for all suffixes of z in a table. Whenever k is invoked with argument v, one looks in the table to see whether the value of $k(v)$ is already there. If so, retnrn that value. If not, compnte $k(v)$ with the given recursion equation, and store the result in the table. This technique, which is known as *exact tabulation* or *memoisation* [11, 31, 54, 70], leads to an efficient program for computing k. Deriving the recursion equation is an essential step in obtaining an efficient computer program. But how do we get from the definition of k to the recursion equation? By dynamic programming! This is the topic of the next section.

1.2 Dynamic Programming

Let ns abstract from the specific example we considered thus far, and concentrate on the following, more general, problem statement:

$$k \;=\; m \circ \exists (\!|h|\!)_F \circ Inv(\!|g|\!)_F \ .$$

All variables in the right-hand side are parameters to this specification. The pattern is the same as in the text formatting example: generate a set of combinatorial objects with $Inv(\!|g|\!)$, evaluate each of these with $(\!|h|\!)$, and pick an optimal value with m. To achieve the generality we want, it will be necessary to interpret k as a relation, because in many optimisation problems, an optimal solution is not unique. In technical terms, the operator m which selects an optimal element from its argument set is a relation and not a function. It is sometimes useful to have g as a general relation, hut h is always functional. For an example where g is not a function, consider the knapsack problem (see e.g. [82]). Here we want to specify the set of all sequences of natural numbers whose sum is below a certain threshold c:

$$t(c) \;=\; \{\, x \mid sum(x) \leqslant c \,\} \ .$$

Clearly, t is the inverse image function of the relation $(\geqslant) \circ sum$, which is not functional. In section 2.2 the knapsack problem will be discussed in more detail.

An appropriate setting for the above requirements is the order-enriched category of relations $\mathit{Rel}(\mathcal{E})$ over a topos $\mathcal{E}$. A topos has precisely the structure needed to interpret the given operators: relations, power transpose and the existential image. There is a small problem, though. How can we apply fold to a relation? What happens to familiar functors and their initial algebras when we consider relations instead of functions? Informally, are the data types of functional programming the same as the data types of relational programming?

The first issue to consider is what happens to functors when we generalise from functions to relations. A regular category $\mathcal{E}$ is included in the category of relations $\mathit{Rel}(\mathcal{E})$ by the *graph functor* G. Say that a functor $H : \mathit{Rel}(\mathcal{D}) \to \mathit{Rel}(\mathcal{E})$ *extends* $F : \mathcal{D} \to \mathcal{E}$ if it agrees with F on functions:

$$HG = GF \ .$$

The following proposition shows that many functors have a unique extension. It relies on our decision to regard $\mathit{Rel}(\mathcal{E})$ as an order-enriched category, which implies that functors on relations are monotonic.

Proposition (Carhoni, Kelly and Wood [23]) *Let $\mathcal{D}$ and $\mathcal{E}$ be regular categories, and $F : \mathcal{D} \to \mathcal{E}$. There exists at most one functor that extends F, which will be denoted F^*. This unique extension exists if and only if F preserves regular epics and F nearly preserves pullbacks.*

The condition that F *nearly preserves pullbacks* may need some further explanation. It means that whenever both

$$\begin{tikzcd} \bullet \arrow[r, "q"] \arrow[d, "p"'] & \bullet \arrow[d, "k"] \\ \bullet \arrow[r, "h"'] & \bullet \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} \bullet \arrow[r, "t"] \arrow[d, "s"'] & \bullet \arrow[d, "F(k)"] \\ \bullet \arrow[r, "F(h)"'] & \bullet \end{tikzcd}$$

are pullback squares, the mediating arrow in

$$\begin{tikzcd} \bullet \arrow[dr, dotted] \arrow[drr, "F(q)"] \arrow[ddr, "F(p)"'] & & \\ & \bullet \arrow[r, "t"] \arrow[d, "s"'] & \bullet \arrow[d, "F(k)"] \\ & \bullet \arrow[r, "F(h)"'] & \bullet \end{tikzcd}$$

is regular epic. One could also say that F nearly preserves pullbacks if and only if F preserves pullbacks *up to image*: in the above diagram (s, t) is the image of $(F(p), F(q))$. From now on, we shall call a functor a *relator* if it preserves regular epics and it nearly preserves pullbacks.

An example of a relator is the existential image functor on a topos, which we already encountered in our example of dynamic programming. On *Set* the existential image functor can be defined by the set comprehension

$$\exists(f)(x) \;=\; \{\, b \mid \exists\, a : a \in x \wedge b = f(a) \,\}\,.$$

In words, f is applied to each element of the set x. Of course, a similar characterisation could be given in the internal lauguage of any other topos. The extension of $\exists : \mathit{Set} \rightarrow \mathit{Set}$ to $\mathit{Rel}(\mathit{Set}) \rightarrow \mathit{Rel}(\mathit{Set})$ cau be characterised by the predicate

$$\begin{aligned} y(\exists^*(R))x \;=\; & \forall\, a \in x : \exists\, b \in y : b(R)a \;\wedge \\ & \forall\, b \in y : \exists\, a \in x : b(R)a\,. \end{aligned}$$

Tbis characterisation also generalises to arbitrary toposes.

When working in a topos (as opposed to a general regular category) it is often quite easy to check whetber a functor is a relator, because one only has to exhibit a so-called *cross-operator*. Cross-operators are a special kind of natural transformation, reminiscent of Beck's distributive laws [8]. Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and F a functor from $\mathcal{D}$ to $\mathcal{E}$. A natural transformation $\gamma : F\exists \rightarrow \exists F$ is said to be a *cross-operator on F* if

1. Crossing singletons gives a singleton,

$$\begin{tikzcd}
F \arrow[d, "F\{\text{-}\}"'] \arrow[dr, "\{\text{-}\}F"] & \\
F\exists \arrow[r, "\gamma"'] & \exists F
\end{tikzcd}$$

2. crossing distributes over union,

$$\begin{tikzcd}
F\exists\exists \arrow[rr, "F\bigcup"] \arrow[d, "\gamma\exists"'] & & F\exists \arrow[d, "\gamma"] \\
\exists F\exists \arrow[r, "\exists\gamma"'] & \exists\exists F \arrow[r, "\bigcup F"'] & \exists F
\end{tikzcd}$$

3. and crossing is monotonic:

$$g \leqslant h \;\Rightarrow\; \gamma(B) \circ F(g) \leqslant \gamma(B) \circ F(h)$$

for all $g, h : A \rightarrow PB$.

How does this tie in with relators? Consider a topos $\mathcal{E}$. The existential image functor induces a monad in $\mathcal{E}$, together with the singleton former and union. It is well-known that the category of relations $\mathit{Rel}(\mathcal{E})$ is isomorphic to the Kleisli-category of the monad $(\exists, \{\text{-}\}, \bigcup)$. Using this isomorphism, we can construct a bijection between extensions of the functor F and cross-operators on F. Together with the result on extending functors, this yields the following proposition:

Proposition *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and $F : \mathcal{D} \rightarrow \mathcal{E}$. There exists at most one cross–operator on F, which is denoted $F\dagger$. This unique cross–operator exists if and only if F is a relator.*

For an example of a cross–operator, consider the product functor

$$\times : (\mathit{Set} \times \mathit{Set}) \rightarrow \mathit{Set} \ .$$

The product functor preserves pullbacks and regular epics, so we know that it has a cross-operator. That cross–operator $(\times)\dagger$ is the natural transformation which takes two sets and returns their cartesian product. Of course, a sequence of sets also has a cartesian product, analogous to $(\times)\dagger$. This induces a cross–operator on the sequence functor, which sends A to the set of finite sequences over A. It follows that the sequence functor is a relator too.

We have now accumulated the apparatus needed to generalise a result of Eilenberg and Wright, which says that a functional initial algebra is also a relational initial algebra. They proved the proposition in a set theoretic context; the introduction of cross–operators makes it possible to reproduce their proof in an arbitrary topos.

Proposition (Eilenberg and Wright [36]) *Let $\mathcal{E}$ be a topos, and $F : \mathcal{E} \rightarrow \mathcal{E}$ a relator with initial algebra $\mu(F)$. Then $\mu(F)$ is also an initial algebra of $F^{\star} : \mathit{Rel}(\mathcal{E}) \rightarrow \mathit{Rel}(\mathcal{E})$.*

For functional programmers, this proposition provides a form of reassurance: they can define their data types in the functional style as before, and use these definitions in program derivations that are conducted in the calculus of relations. Together with the result on extending functors, it justifies the slogan

> *"The generalisation of programming with total functions to programming with relations is free of surprises."*

This completes the summary of results needed to formulate a theory of dynamic programming. Here is the main theorem:

Theorem *Let $\mathcal{E}$ be a topos, and consider the following equation in $\mathit{Rel}(\mathcal{E})$:*

$$k \;=\; m \circ \exists\,(\![h]\!)_{F^*} \circ \mathit{Inv}(\![g]\!)_{F^*}\;.$$

Suppose that

1. *$F : \mathcal{E} \to \mathcal{E}$ is a relator that has an initial algebra,*
2. *$h : F(A) \to A$ is a function,*
3. *the relation $m : P(A) \to A$ satisfies*

$$m \circ \{\text{-}\}(A) \;=\; 1(A) \;\; \textit{and} \;\; m \circ \bigcup(A) \;=\; m \circ \exists^*(m) \quad \text{and}$$

4. *h distributes over m in the following sense*

$$\begin{tikzcd}
FP(A) \arrow[rr, "F^*(m)"] \arrow[d, "F\dagger(A)"'] & & F(A) \arrow[d, "h"] \\
PF(A) \arrow[r, "\exists(h)"'] & P(A) \arrow[r, "m"'] \arrow[u, phantom, "\boxed{?}"] & A
\end{tikzcd}$$

where $\boxed{?}$ is $(\supseteq)$, $(\subseteq)$ or $(=)$.

Then k satisfies

$$\begin{tikzcd}
B \arrow[r, "\mathit{Inv}(g)"] \arrow[d, "k"'] \arrow[dr, phantom, "\boxed{?}"] & PF(B) \arrow[d, "\exists^*(h \circ F^*(k))"] \\
A & P(A) \arrow[l, "m"]
\end{tikzcd}$$

Let us carefully step through the theorem, using the text formatting example to clarify its applicability conditions. First, the definition of k. The relation k is defined as an instance of the generic specification of dynamic programming. The inverse image $Inv(\![g]\!)$ is a function that generates a set of combinatorial objects. In our example, this is the set of all layouts. The existential image $\exists\,(\![h]\!)$ assigns a value to each element of the set; in the example that is the amount of white space in a paragraph. Finally, m is a non–deterministic mapping that selects an optimal element from its argument set. In the example m is the function that returns the minimum of its argument; the minimum of the empty set is infinity.

The first applicability condition concerns the functor F. The functor F should be a relator, so it can be extended to relations. Furthermore it should have an initial algebra — otherwise it does not describe a data type. Later on we shall see that so–called *polynomial* functors satisfy this criterion. In particular, the functor that defines the data type of paragraphs is polynomial, and therefore it is a relator that has an initial algebra.

The second applicability condition says that h is a *functional* F–algebra, which means that it lies in the image of the graph functor G. Strictly speaking, it is possible to formulate a slightly more general theorem without this assumption, but the added generality does not seem to be useful. The assumption that h is functional will be essential in later sections, when we develop corollaries to the above theorem.

The relation $m : P(A) \rightarrow A$ should be a *selector*, that is, selecting an element from a singleton gives the single element. Furthermore, if one selects an optimal element from the union of a collection of sets, one may as well select an optimal element from each of the component sets, and then select an optimal element from those optima. It might appear that m is a selector iff it is an object in the Eilenberg–Moore category of $(\exists^*, \{\cdot\}, \bigcup)$. There is a catch, however: $(\exists^*, \{\cdot\}, \bigcup)$ is only a lax monad in $Rel(\mathcal{E})$, and so one cannot talk about its Eilenberg-Moore category in the usual sense. In the next section, we shall study selectors in more detail.

The last condition of the theorem, that h distributes over m, is the really interesting one. It is a formal statement of the principle of optimality, which is the property that an optimal solution is composed of optimal solutions to subproblems. Let us try to understand that property in terms of our running example. Take equality $(=)$ for $\boxed{?}$. When we express the commuting diagram in conventional set theory, it reads

$$min\ \{\, f(l) + n \mid n \in x \,\} \quad = \quad f(l) + min\,(x)\ .$$

This identity is satisfied because addition is monotonic.

It remains to discuss the conclusion of the theorem,

$$k \; = \; m \circ \exists^{\star}(h \circ F^{\star}(k)) \circ \mathit{Inv}(g) \; .$$

When interpreted in an operational manner, this recursion equation is in line with the traditional presentation of dynamic programming that one finds in textbooks on algorithm design, *e.g.* [82]. First, decompose the argument in all possible ways with the inverse image function of g, ($\mathit{Inv}(g)$). This generates a set of decompositions, each of which is solved recursively with $F^{\star}(k)$. The solutions to subproblems are then composed by the F–algebra h into solutions for the whole problem. Hence, one could say that

$$\exists^{\star}(h \circ F^{\star}(k)) \circ \mathit{Inv}(g)$$

generates a set of candidate solutions. The relation m selects an optimal element from this set.

It should be mentioned that the theorem is not always useful in deriving programs from specifications, even when it is applicable. Indeed, sometimes one obtains a recursion equation which is not an algorithm at all, because $\mathit{Inv}(g)$ splits an argument x into x itself (and possibly something else). In such cases the recursion 'does not make progress', and the theorem is useless. In the text formatting example, this problem was avoided by excluding the possibility of empty lines in a paragraph.

1.3 Constructing Selectors from Preorders

One of the applicability conditions of the theorem about dynamic programming is that $m : P(A) \rightarrow A$ should be a *selector*.

$$m \circ \{\cdot\}(A) \; = \; 1(A) \quad \text{and} \quad m \circ \bigcup(A) \; = \; m \circ \exists^{\star}(m) \; .$$

This condition is very difficult to meet. To see how stringent it is, consider the following counter–example. Let A be a non–empty set. Given a preorder $R : A \rightarrow A$, the relation $min(R)$ which maps its argument set to its minimum elements is *not* a selector, because the empty subset of A does not have a minimum element. As a consequence, we have the inclusion

$$min(R) \circ \exists^{\star}(min(R)) \; \subseteq \; min(R) \circ \bigcup(A) \; ,$$

but not inclusion the other way round. It is for this reason that the fictitious value *infinity* (∞) was introduced in the example specification: the minimum element of the empty set is infinite. This trick of introducing infinity seems ad hoc, and in this section we shall discuss a more systematic approach to constructing selectors from preorders.

To start with, let us formnlate a precise definition of the relation $min(R)$. In the category of relations over a topos, one can take the *right-quotient* $R\backslash S$ of two relations $R : A \rightarrow B$ and $S : A \rightarrow C$ with a common source A. This division operator $(\backslash)$ is characterised by the following equivalence:

$$T \subseteq R\backslash S \quad \text{iff} \quad T \circ S \subseteq R \,.$$

In words, $R\backslash S$ is the maximnm morphism T that makes the triangle

$$\begin{tikzcd}
A \arrow[dr, "S"] \arrow[dd, "R"'] & \\
\supseteq & C \arrow[dl, "T"] \\
B &
\end{tikzcd}$$

semi-commute.

Given an endorelation R on A, one can define the relation $min(R) : P(A) \rightarrow A$ by

$$min(R) \;=\; R\backslash{\in^\circ}(A) \;\cap\; {\in}(A) \,.$$

This is in accordance with the usual set theoretic definition: a is a minimum element of x if it is a lower bound of x and it is an elemeut of x.

The *domain* of a relation is defined as follows:

$$Dom(R : A \rightarrow B) \;=\; 1(A) \cap (R^\circ \circ R) \,.$$

This also corresponds to the usual definition in set theory: a is in the domain of R if there exists an element b such that R relates a to b. A relation $R : A \rightarrow B$ is said to be *entire* if the domain of R coincides with the identity on A. (Some authors call entire relations *total* or *everywhere defined.*) The relation $min(R)$ is *not* a selector, because the empty set is not in its domain. In a set theoretical context, we know how to make $min(R)$ entire, namely by adding a fictitious value (infinity) to its target. Fnrthermore, the introdnction of infinity makes it possible to turn $min(R)$ into a selector. Can we generalise that construction to arbitrary toposes? Does there exist a canonical way of making a relation entire?

In a topos, partial arrows can be made entire in a canonical way because the embedding of a topos into its category of partial arrows has a right adjoint. Do we also have an adjunction between the category of entire relations and the category of relations? Even for *Set*, the answer is no: there exists a weak universal arrow, hut it is not proper because there are many different ways of making a relation entire.

Apparently, to generalise the canonical construction in *Set*, we need a different approach.

The unit of the adjunction between a topos and its partial arrows is sometimes called a *partial arrow classifier*. We shall generalise this concept to the notion of a *relation totaliser* by adding an extra condition that yields the desired uniqueness.

Consider a regular category $\mathcal{E}$. A *relation totaliser* is a collection of monics in $\mathcal{E}$ with the following universal property: for each relation $R : A \rightarrow B$ there exists precisely one entire relation $\tilde{R} : A \rightarrow \tilde{B}$ such that

$$\eta^\circ(B) \circ \tilde{R} \;=\; R \quad\text{and}\quad \tilde{R} \circ Dom(R) \;\subseteq\; \eta(B) \circ R\ .$$

In *Set*, $\tilde{B}$ is the set B augmented with a fictitious value; the arrow $\eta(B)$ is the embeddiug of B into $\tilde{B}$. The first equation says that if you forget about the fictitious element in $\tilde{B}$, you get R back. The second equation says that if a is in the domain of R, $\tilde{R}$ relates it to the same elements of B as R does.

It would be nice if every topos had a relation totaliser, but this is not true. A counter–example is $Set^{\rightarrow}$, the category of commuting squares. For the purpose of this thesis, however, the following result suffices:

Proposition *In a Boolean topos, the partial arrow classifier is also a relation totaliser.*

Let us return to the prohlem of constructing selectors from preorders. Given a relation $R : A \rightarrow A$ one can define the *selector* of R, denoted $sel(R) : P(\tilde{A}) \rightarrow \tilde{A}$, by the following equation:

$$sel(R) \;=\; \widetilde{min}(R) \circ \exists(\eta^\circ(A))\ .$$

The existential image $\exists(\eta^\circ(A))$ removes all fictitious values from its argument set. The relation $\widetilde{min}(R)$ returns a minimnm element of the resulting set, provided such a minimum element exists. If it does not exist, $\widetilde{min}(R)$ yields a fictitious value. Assuming that R is a well–founded preorder, $sel(R)$ is indeed a selector:

Proposition *Let $\mathcal{E}$ be a Boolean topos. Let $R : A \rightarrow A$ be a well–founded preorder in $Rel(\mathcal{E})$,* i.e.

$$1(A) \subseteq R\ , \quad R \circ R \subseteq R\ , \text{ and } \quad Dom(\in(A)) \subseteq Dom(min(R))\ .$$

Then $sel(R)$ is a selector.

For example, the minimum function $min : P(\mathrm{N} \cup \{\infty\}) \rightarrow (\mathrm{N} \cup \{\infty\})$ can be defined as the selector of the standard ordering on natural numbers.

1.4 Monotonicity implies Distributivity

The above construction of selectors covers many optimisation problems that occur in practice. It is worthwhile, therefore, to see whether the other applicability conditions of dynamic programming can he simplified when the selector is of the form $sel(R)$. Recall the formal statement of the principle of optimality:

$$\begin{tikzcd}
FP(A) \arrow[rr, "F^*(sel(R))"] \arrow[dd, "F\dagger(A)"'] & & F(A) \arrow[dd, "h"] \\
& \boxed{?} & \\
PF(A) \arrow[r, "\exists(h)"'] & P(A) \arrow[r, "sel(R)"'] & A
\end{tikzcd}$$

where $\boxed{?}$ is inclusion ($\subseteq$), containment ($\supseteq$) or equality ($=$). In the text formatting example, it was claimed that the principle of optimality is satisfied because additiou is monotonic. This seems to be the general pattern: to verify that h distributes over $sel(R)$, it suffices to show tbat h is iu some sense monotonic with respect to the preorder R. To make this intuition precise, we shall need some further restrictions on the functor F.

Let $\mathcal{E}$ be a category with products and coproducts. The class of *polynomial* endofunctors on $\mathcal{E}$ is inductively defined by the following clauses:

1. The identity functor on $\mathcal{E}$ is polynomial.
2. If A is an object of $\mathcal{E}$, the constant functor which maps all arrows to the identity on A is polynomial.
3. If G and H are polynomial functors, then $G\hat{+}H$ and $G\hat{\times}H$ defined by

$$\begin{aligned}(G\hat{+}H)(k) &= G(k)+H(k)\\ (G\hat{\times}H)(k) &= G(k)\times H(k)\end{aligned}$$

   are also polynomial.

The class of polynomial functors is of course very restricted, but it suffices to define the simple data types encountered in dynamic programming. Polynomial functors enjoy a number of properties that are useful in the present context. The first property says that they extend uniquely to relations:

Proposition *In a topos, polynomial functors are relators.*

This proposition follows from the fact that in a topos coproducts preserve pullbacks, and the coproduct injections are disjoint. The second property says that any polynomial functor describes a data type:

Proposition (Johnstone [56]) *In a topos with a natural numbers object, polynomial functors have initial algebras.*

The fact that polynomial functors are relators, and other, more technical properties, yield that monotonicity implies distributivity:

Proposition *Let $\mathcal{E}$ be a Boolean topos, and let F be a polyuomial endofunctor on $\mathcal{E}$. Let $h : F(A) \rightarrow A$ be a functional F-algebra, and let R be well-founded preorder on A. Finally, let $\boxed{?}$ be $(\subseteq)$, $(\supseteq)$ or $(=)$. If h is monotonic with respect to R,*

$$\begin{tikzcd}
A \arrow[d, "R"'] & F(A) \arrow[l, "h"'] \arrow[d, "F^{*}(R)"] \arrow[dl, phantom, "\boxed{?}"] \\
A \arrow[r, "h^{\circ}"'] & F(A)
\end{tikzcd}$$

then h distributes over $sel(R)$:

$$\begin{tikzcd}
PF(\tilde{A}) \arrow[d, "\exists(E(h) \circ \overline{F}(A))"'] & FP(\tilde{A}) \arrow[l, "F\dagger(\tilde{A})"'] \arrow[r, "F^{\star}(sel(R))"] \arrow[d, phantom, "\boxed{?}"] & F(\bar{A}) \arrow[d, "\overline{F}(A)"] \\
P(\tilde{A}) \arrow[r, "sel(R)"'] & \bar{A} & \widetilde{F(A)} \arrow[l, "E(h)"]
\end{tikzcd}$$

where $\overline{F}(A) = ((F(\eta(A)))^{\circ})^{\sim}$ and $E(h) = (h \circ \eta_A^{\circ})^{\sim}$.

A few comments about the operators that are defined in this proposition. The functor E is the endofunctor on $\mathcal{E}$ that comes with the partial arrow classifier. The collection of arrows $\overline{F}(A)$ forms a natural transformation $F \circ E \rightarrow E \circ F$. It is in fact the cross-operator $F\dagger$, restricted to sets that have at most one element. Intuitively $\overline{F}(A)$ takes a structure, say a tuple, and it maps that tuple to the fictitious element if one of the components is fictitious. If all elements in the tuple are proper values, it just returns the tuple. Here its rôle is to make h strict with respect to the fictitious value: $E(h) \circ \overline{F}(A)$ is the strict version of h.

1.5 Summary

What has been achieved so far? First we provided a precise characterisation of dynamic programming. From a purely theoretical perspective, that result is satisfactory: it is general and elegant. In practice, however, the applicability conditions may be hard to check. Motivated by this observation, we introduced the construction of selectors from preorders. Subsequently, that construction was used to simplify the principle of optimality. The earlier formulation of dynamic programming can therefore be replaced by the following statement:

Theorem *Let $\mathcal{E}$ be a Boolean topos with a natural numbers object. Let F be a polynomial endofunctor on $\mathcal{E}$, let $h : F(A) \to A$ be a functional F-algebra, and $g : F(B) \to B$ a relational F^*-algebra. Let R be a well-founded preorder on A. Define $k : B \to \tilde{A}$ by*

$$k \;=\; (min(R) \circ \exists\,(\![h]\!) \circ Inv((\![g]\!)))^{\smile} \,.$$

Let $\boxed{?}$ be $(\subseteq)$, $(\supseteq)$ or $(=)$. If h is monotonic with respect to R,

$$\begin{tikzcd}
A \arrow[d, "R"'] \arrow[dr, phantom, "\boxed{?}"] & F(A) \arrow[l, "h"'] \arrow[d, "F^*(R)"] \\
A \arrow[r, "h^\circ"'] & F(A)
\end{tikzcd}$$

then k satisfies

$$\begin{tikzcd}
B \arrow[r, "Inv(g)"] \arrow[d, "k"'] \arrow[dr, phantom, "\boxed{?}"] & PF(B) \arrow[d, "\exists^*(t)"] \\
\tilde{A} & P(\tilde{A}) \arrow[l, "sel(R)"]
\end{tikzcd}$$

where t is the composite

$$F(B) \xrightarrow{F^*(k)} F(\tilde{A}) \xrightarrow{\overline{F}(A)} \widetilde{F(A)} \xrightarrow{E(h)} \tilde{A}$$

To appreciate the practical significance of this new result, let us briefly return to the text formatting example. We only considered how the minimum amount of white space in a layont may be computed, not how such an optimal layout itself may be constructed. There are two reasons why the *value* problem was more easily presented than the *object* problem. First, in the object problem one has to introduce an object (a paragraph) of infinite cost. This is intuitively more difficult to accept than an infinite natnral number, although our treatment in terms of relation totalisers shows that both arise by the same construction. A second difficulty in reasoning about the object problem is the presence of arbitrary relations instead of just (partial) functions. With functions, we are on familiar ground and we can appeal to our intuition for their basic properties, but not so for relations. And yet, it seems that the object problem and the value problem are in some sense equivalent: if dynamic programming is applicable to either of them, it is also applicable to the other.

Let us try to make this a bit more precise. Given F-algebras g and h, and a preorder R, the *value* problem is given by

$$v \;=\; (\mathit{min}(R) \circ \exists\, (\!|h|\!) \circ \mathit{Inv}(\!|g|\!))^{\urcorner} \,.$$

If R is a partial order, v is a total function. This is not the case for the *object* problem, which is specified as

$$o \;=\; (\mathit{min}((\!|h|\!)^{\circ} \circ R \circ (\!|h|\!)) \circ \mathit{Inv}(\!|g|\!))^{\urcorner} \,.$$

Note that the existential image which is explicit in the generic specification is implicit in the object problem, for

$$1(P(T)) \;=\; \exists\, 1(T) \;=\; \exists\, (\!|\mu(F)|\!) \,,$$

where $\mu(F) : F(T) \rightarrow T$ is the initial F-algebra. Clearly, the value problem can be reduced to the object problem, because the triangle

$$\begin{tikzcd}
B \arrow[rr, "v"] \arrow[dr, "o"'] & & \tilde{A} \\
& \tilde{T} \arrow[ur, "E(\!|h|\!)"'] &
\end{tikzcd}$$

commutes. It is not so clear, however, that a dyuamic programming solution for the value problem also yields a dynamic programming solution for the object problem. The next proposition goes some way towards solving that difficulty. It says that if the monotonicity condition of dynamic programming is met for the value problem, it is also met for the objert problem. Hence, our new formulation of dynamic programming not only simplifies the applicability conditions: it also clarifies the relationship between value and object problems.

Proposition *Let $\mathcal{E}$ be a regular category, and let $F : \mathcal{E} \to \mathcal{E}$ be a relator that has initial algebra $\mu(F) : F(T) \to T$. Furthermore, let $h : F(A) \to A$ be an F-algebra. Let $\boxed{?}$ be $(\subseteq)$, $(\supseteq)$ or $(=)$. Then*

$$\begin{tikzcd}
A \arrow[d, "R"'] \arrow[dr, phantom, "\boxed{?}"] & F(A) \arrow[l, "h"'] \arrow[d, "F^\star(R)"] & \text{implies} & T \arrow[d, "S"'] \arrow[dr, phantom, "\boxed{?}"] & F(T) \arrow[l, "\mu(F)"'] \arrow[d, "F^\star(S)"] \\
A \arrow[r, "h^\circ"'] & F(A) & & T \arrow[r, "(\mu(F))^\circ"'] & F(T)
\end{tikzcd}$$

where $S = (\![h]\!)^\circ \circ R \circ (\![h]\!)$.

In particular, dynamic programming is applicable to the object problem in the text formatting example because addition is monotonic.

2 Applications

2.1 String–to–String Correction

Given are two strings of characters, say x and y. The aim is to transform each string into the other by performing a sequence of edit operations on both of them together. There are three edit operations available to achieve this task, named *movexy*, *moveyx* and *swap*. Their intended meanings are stated below:

movexy a	move character a from x to y
moveyx b	move character b from y to x
swap (a, b)	swap a in x with b in y

These operations are applied while traversing x and y from left to right. For example, let x = "sediment" and y = "eldritch". Two sequences of edit operations that transform x and y into each other are the following:

edit operation		x	y
movexy	s	s	
swap	(e, e)	e	e
moveyx	l		l
swap	(d, d)	d	d
moveyx	r		r
swap	(i, i)	i	i
movexy	m	m	
movexy	e	e	
movexy	n	n	
swap	(t, t)	t	t
moveyx	c		c
moveyx	h		h

edit operation		x	y
movexy	s	s	
swap	(e, e)	e	e
swap	(d, l)	d	l
swap	(i, d)	i	d
swap	(m, r)	m	r
swap	(e, i)	e	i
swap	(n, t)	n	t
swap	(t, c)	t	c
moveyx	h		h

A sequence of edit operations that transforms x and y into each other is said to be a ***transform of*** x ***and*** y. Both of the above examples are transforms of "sediment" and "eldritch".

Associated with each edit operation is a cost, given by the function w. A possible definition of w is the following:

$$\begin{array}{rcl} w\,(\mathit{movexy}\ a) & = & 1 \\ w\,(\mathit{moveyx}\ b) & = & 1 \\ w\,(\mathit{swap}\ (a,b)) & = & \begin{cases} \infty & \text{if} \quad a \neq b \\ 0 & \text{if} \quad a = b \end{cases} \end{array}$$

The total cost of a sequence of edit operations is the sum of the costs of its elements. Consider for example the transforms listed above. With the suggested choice of w, the transform on the left-hand side has cost 8 while the transform on the right-hand side is of infinite cost. The purpose of this section is to construct an algorithm that yields a transform of x and y of minimum cost. This programming exercise is known as the *string-to-string correction problem*, and it was originally studied by Wagner and Fischer in [97]. Some variants of the problem are of practical interest; a recent paper by Apostolico *et al.* [2] cites widely divergent applications in speech recognition, machine vision and molecular sequence comparison.

Note that the problem of determining a longest common subsequence of x and y is a special case of the string-to-string correction problem. With the above choice of w, an optimal transform swaps only identical characters, and it contains a longest common subsequence of x and y in the list of *swap* operations. Let us check the validity of this claim in the above example. The transform on the left-hand side is the unique transform of minimum cost. The longest common subsequence of "sediment" and "eldritch" is "edit"; this is precisely the sequence of characters in the list of *swap* operations.

As in the text formatting example, the first step towards a formal specification is to define the relevant types. The string-to-string correction problem involves two kinds of lists: lists of characters (the input) and lists of edit operations (the output). It is worthwhile, therefore, to give a definition of lists that is parametrised by the element type. The lists considered here are finite, possibly empty lists. The lists over a set of elements E can be viewed as an initial F_E-algebra by defining

$$F_E(A) \quad = \quad \top + (E \times A)$$

where $\top$ is the terminal object of *Set*. From now on, E^* stands for the set of lists over E, which is the target of the initial F_E-algebra. Furthermore, we shall denote the initial F_E-algebra itself by

$$[Nil, (:)] : (\top + E \times E^*) \to E^* \ .$$

Here *Nil* is the constant function returning the empty list. The binary operator $(:)$ takes an element and a list, and places the element at the front of the list.

The data type of edit operations is modelled by the coproduct

$$O \;=\; A + A + (A \times A) \;,$$

where A is the set of characters. In line with the informal problem description, we shall write *movexy*, *moveyx*, and *swap* for the respective coproduct injections. The data type of sequences of edit operations is O^*.

The function $eval : O^* \rightarrow (A^* \times A^*)$ takes a sequence of edit operations and yields two strings hy performing the prescribed edit actions:

$$eval \;=\; (\![[(Nil, Nil), \oplus]]\!)$$

where $\oplus : O \times (A^* \times A^*) \rightarrow (A^* \times A^*)$ is given by

$$\begin{array}{rcl} movexy(a) \oplus (u,v) & = & (u, a : v) \\ moveyx(b) \oplus (u,v) & = & (b : u, v) \\ swap(a,b) \oplus (u,v) & = & (b : u, a : v) \;. \end{array}$$

In the informal problem description, we saw two examples of seqnences of edit operations. Applied to either of these sequences, *eval* yields the pair

$$(\text{"eldritch"}, \text{"sediment"}) \;.$$

Let us say that a sequence z is a *transform* of x and y if $eval(z) = (y, x)$. Clearly, the set of all transforms of x and y is returned by

$$Inv(eval)(y,x) \;=\; \{\, z \mid eval(z) = (y,x) \,\} \;,$$

the inverse image fuuction of *eval*.

Recall the goal of this programming exercise: to find a transform of minimum cost. We have just formulated a precise definition of transforms; the next step is to define the cost function c. The cost of a sequence of edit operations is the sum of the costs of its elements:

$$c \;=\; (\![[0, \otimes]]\!)$$

where $a \otimes n = w(a) + n$.

The string–to–string correction problem can now be formulated as an instance of the generic specification of dynamic programming. Let k stand for the relation that takes a pair of strings (y, x), and retnrns a transform of x and y of minimum cost:

$$k \;=\; (\, min(c^\circ \circ (\leq) \circ c) \;\circ\; Inv(eval)\,)^\smile \,.$$

Again dynamic programming is applicable because addition is monotonic. When you unfold the abstract definitions in the theorem about dynamic programming, you obtain the original algorithm of Wagner and Fischer [97]. In this presentation, we have confined our attention to the simplest form of string–to–string correction, but the same idea works for more complicated variants, e.g. the *modified edit distance* problem discussed by Galil and Giancarlo in [41].

2.2 Loading

A vessel is to be loaded with containers. The containers that may be selected for sbipping are lined up on the quay, and associated witb each container is a value and a weight, which are both natural numbers. The total weight of a cargo is the sum of the weights of the containers selected. Likewise, the value of a cargo is the sum of its parts. The vessel can carry a cargo of limited weight only, and the problem is therefore to maximise the total value of a cargo, without exceeding the carrying capacity of the vessel.

This programming exercise is known as the *knapsack* problem — here we have chosen lo load a vessel rather than to pack the traditional knapsack. The aim in presenting this example is to demonstrate why we chose to allow g to be a relational $F^\star$–algebra in the generic specification

$$(\, sel(R) \circ \exists\, (\!|h|\!) \circ Inv(\!|g|\!)\,)^{\smile} \ ,$$

and not just a functional F–algebra, as h must be.

The line of containers on the quay will be modelled as a list of

$$(\,\text{value, weight}\,)$$

pairs. Both the value and the weight of a container are natural numbers, and the set C of all containers is the cartesian product $\mathrm{N} \times \mathrm{N}$. For ease of reference, we name the projection functions from C to the natural numbers v (for *value*) and w (for *weight*) respectively:

$$v\,(a,b) \;=\; a \quad \text{and} \quad w\,(a,b) \;=\; b\ .$$

Let $z \in C^*$ be the line of containers that are awaiting shipment on the quay. To select a cargo from z, each container will be labelled with either 0 or 1. A label '0' means that the container is not selected, whereas a label '1' means that it will be part of the cargo. A cargo is therefore a hist of pairs, an element of

$$(C \times \{0,1\})^*\ .$$

Not every selection of containers $s \in (C \times \{0,1\})^*$ is a possible cargo; certain constraints have to be satisfied. Firstly, s should be a selection of containers from z. This requirement may be expressed as follows:

$$\mathit{untag}(s) \;=\; z \;,$$

where *untag* is the function that removes the labels:

$$\mathit{untag}\;[(n_1,b_1),(n_2,b_2),\ldots(n_m,b_m)] \;=\; [n_1,n_2,\ldots,n_m]\;.$$

Secondly, the total weight of the containers selected should not exceed the carrying capacity of the vessel. Let $c \in N$ be that carrying capacity. To be a possible cargo, s should satisfy the inequation

$$\mathit{weight}(s) \;\leq\; c\;,$$

where *weight* is the function that returns the total weight of a selection of containers:

$$\mathit{weight}\;[(n_1,b_1),(n_2,b_2),\ldots,(n_m,b_m)] \;=\; \sum_{i=1}^{m}(w(n_i)\times b_i)\;.$$

The next step towards a formal specification of the loading problem is to define the function *cargos* that takes a list z and a capacity c as parameters, and returns the set of all possible cargos. It is given by the set comprehension below:

$$\mathit{eargos}\;(z,c) \;=\; \{\, s \in (C \times \{0,1\})^* \mid \mathit{untag}(s) = z \;\wedge\; \mathit{weight}(s) \leq c \}$$

Motivated by a wish to apply dynamic programming, we aim to express *cargos* as an inverse image function. To do so, we need a generalisation of the product in $\mathcal{E}$ to $\mathit{Rel}(\mathcal{E})$. For $R : A \rightarrow B$ and $S : A \rightarrow C$ relations with a common source A, define

$$\langle R,S\rangle^* \;=\; (R \times^* S) \circ \langle 1(A), 1(A)\rangle \;:\; A \rightarrow (B \times C)\;.$$

(If R and S lie in $\mathcal{E}$, this is the ordinary product.) By some tedious manipulations using initiality of lists and monotonicity of addition, one may derive that

$$\mathit{cargos} \;=\; \mathit{Inv}(\!|[\langle \mathit{Nil}, (\geq)\circ 0\rangle^*, \odot]|\!)\;,$$

where $\odot$ is the function defined by

$$(n,b)\odot(z,c) \;=\; (n:z,\;(w(n)\times b)+c)\;.$$

The value of a cargo is the sum of the values of the containers selected. The function that takes a cargo and returns its value is called *value*. It is easily defined using fold:

$$\mathit{value} \;=\; (\!|[0,\oslash]|\!)\;,$$

where $(n,b) \oslash m = (v(n) \times b) + m$.

The loading problem can now be expressed as an instance of the generic specification:

$$k \;=\; (\, \mathit{min}(\geq) \circ \exists(\mathit{value}) \circ \mathit{cargos}\,)^{\smile} \;.$$

There is a small problem in the application of dynamic programming, becanse the converse order on numbers is not well–founded: an infinite set of natural numbers does not have a maximum element. This problem may be solved by restricting the values of a cargo to a finite set $\{0,1,\ldots,m\}$, and stipulating that m is a zero of addition. For this modified specification, our theorem about dynamic programming is applicable because addition is monotonic.

By mechanically instantiating our result about dynamic programming and using some elementary arithmetic, one obtains the following equations for k:

$$\begin{array}{lcl} k(\mathit{Nil},c) & = & 0 \\ k(n:z,c) & = & \left\{ \begin{array}{ll} k(z,c) & \text{if } \; w(n) > c \\ k(z,c) \sqcup (v(n) + k(z, c - w(n))) & \text{if } \; w(n) \leq c \;. \end{array} \right. \end{array}$$

Compare this recnrsion to the text formatting algorithm, which we discnssed at the beginning of this thesis:

$$k(z) \;=\; t(z) \sqcap (\mathit{min}\; \{\, f(u) + k(v) \mid u +\!\!\!+\, v = z \,\}) \;.$$

Expressed in their traditional form the two algorithms look very different, and yet they are abstractly the same.

2.3 Bracketing

Consider the type of binary trees that have data only at their leaves. Such trees are fully parenthesised representations of their frontier, and therefore they are called *bracketings*. In this section, we shall be concerned with constructing a bracketing of minimum cost for a given frontier. This problem arises, for example, when multiplying a sequence of matrices: different bracketings may lead to vast differences in the number of arithmetical operations required to perform the multiplication. It will be shown that the problem of hracketing a sequence of matrices for mnltiplication does not satisfy the generic specification of dynamic programming. We shall analyse the difficulty, and present a generic solution which is also applicable to other programming problems, like the constrnction of optimal binary search trees.

As in the preceding examples we start by defining the types. There are three types involved in the hracketing problem: the type of matrices, the type of bracketings and the type of non–empty lists, which we have already seen in the text formatting example.

Let M be the type of matrices. The function $rows : M \to \mathrm{N}$ returns the number of rows in a matrix; similarly, $cols : M \to \mathrm{N}$ yields the number of columns. These two operators distribute through matrix multiplication in the following sense:

$$rows(m * n) \;=\; rows(m) \;\text{ and }\; cols(m * n) \;=\; cols(n)\;.$$

The data type of *bracketings* over M is defined to be the initial F–algebra, where the functor $F : Set \to Set$ is given by

$$F(A) \;=\; M + (A \times A)\;.$$

It will be expedient to have an explicit name for the set of all bracketings, and we shall call it B. Since the initial F–algebra is a coproduct, one can name its individual components:

$$[\widehat{\cdot}, \pm] \;=\; \mu(F)\;.$$

The first component of this coproduct ($\widehat{\cdot}$) takes a matrix and turus it into a singleton tree. The second component ($\pm$) is a binary operator that joins two trees together. Hence one might say that

$$((m_1\,(m_2\,m_3))\,(m_4\,(m_5\,m_6)))$$

is shorthand for

$$((\widehat{m_1} \pm (\widehat{m_2} \pm \widehat{m_3})) \pm (\widehat{m_4} \pm (\widehat{m_5} \pm \widehat{m_6})))\;.$$

Now that bracketings have been properly defined, we can make the notions *frontier* and *bracketing of a frontier* more precise. As before, $+\!\!+$ denotes the associative operator that concatenates two sequences. Singleton sequences are constructed by $[\cdot]$, and thus we have for instance

$$[m_1, m_2, m_3] \;=\; [m_1] +\!\!+ [m_2] +\!\!+ [m_3]\;.$$

The fold $(\![[\cdot], +\!\!+]\!)$ takes a bracketing and flattens it into a sequence, which is said to be the *frontier* of the bracketing. Here is an example of a frontier:

$$\begin{array}{ll} & frontier((m_1\,(m_2\,m_3))\,(m_4\,(m_5\,m_6))) \\ = & \\ & [m_1, m_2, m_3, m_4, m_5, m_6]\;. \end{array}$$

Given a sequence x, we are interested iu the set of all hracketings that have x as a frontier. This set is returned by the inverse image $Inv(frontier)$, since we have the following characterisation of the inverse image in Set:

$$Inv(frontier)(x) \;=\; \{\, t \in B \mid frontier(t) = x \,\}\;.$$

Next, we need to define the cost of a bracketing. The cost of a bracketing of a matrix product is the number of scalar multiplications required in its evaluation. For simplicity we assume that the naive matrix multiplication algorithm is used, so the cost of compnting $n * p$ is

$$\mathit{rows}\,(n) \times \mathit{rows}\,(p) \times \mathit{cols}\,(p)\ .$$

The cost of a bracketing may be defined as follows. The cost of a singleton tree is zero, since no matrix multiplication needs to be performed. The cost of a composite tree $(t_0 \pm t_1)$ is the cost of evaluating the products of t_0 and t_1, plus the cost of multiplying these two products. Note that the final term only depends on the frontier x_0 of t_0 and the frontier x_1 of t_1, because matrix multiplication is associative. We shall write $w(x_0, x_1)$ for the cost of multiplying the product of a frontier x_0 with the product of a frontier x_1. Formally, the cost function $c : B \rightarrow \mathbb{N}$ is given by the equations below:

$$\begin{aligned} c(\widehat{m}) &= 0 \\ c(t_0 \pm t_1) &= c(t_0) + c(t_1) + w(x_0, x_1) \end{aligned}$$

where x_0 and x_1 are the frontiers of t_0 and t_1 respectively. It remains to give a formal definition of the function w:

$$\begin{aligned} & w\,(\,[n_1, n_2, \ldots, n_r],\, [m_1, m_2, \ldots, m_s]\,) \\ = \; & \\ & \mathit{rows}(n_1)\ \times\ \mathit{rows}(m_1)\ \times\ \mathit{cols}\,(m_s) \end{aligned}$$

This definition requires some further explanation. Consider the product $(\prod m_i)$ of the matrices $m_1, m_2, \ldots, m_s$. The uumber of rows in $(\prod m_i)$ equals the number of rows in m_1:

$$\mathit{rows}\,(\prod m_i) \;=\; \mathit{rows}(m_1)\ .$$

Similarly, we have for the columns $\mathit{cols}\,(\prod m_i) = \mathit{cols}\,(m_s)$. It follows that the number of scalar multiplications required to multiply the matrix products $(\prod n_i)$ and $(\prod m_i)$ is given by

$$\mathit{rows}(n_1)\ \times\ \mathit{rows}(m_1)\ \times\ \mathit{cols}\,(m_s)\ .$$

We now aim to express the bracketing problem as an instance of dynamic programming. Recall that the inverse image function of *frontier* returns all bracketings of its argument. The cost function c assigns to each bracketing of a matrix product the cost of evaluating that bracketing. The minimum cost of a bracketing is therefore specified by the relation k given below:

$$k \;=\; (\ \mathit{min}(\leq) \circ \exists(c) \circ \mathit{Inv}(\mathit{frontier})\,)^{\smile}\ .$$

This expression does not match the generic specification of dynamic programming, becanse the cost function c is not a fold. At first sight this may seem alarming, for the bracketing problem is considered a typical application of dynamic programming. As we shall see shortly, however, there exists a simple solution to this difficulty.

The function c is not a fold because there exists no function l that makes

$$\begin{tikzcd}
M + B^2 \arrow[r, "{[\hat{-}, \pm]}"] \arrow[d, "1 + c^2"'] & B \arrow[d, "c"] \\
M + \mathrm{N}^2 \arrow[r, "l"'] & \mathrm{N}
\end{tikzcd}$$

commute. It stands to reason, therefore, that we look for a similar property that is not quite so stringent. Using the definitions of F and c above, it can be shown that

$$\begin{tikzcd}
M + B^2 \arrow[r, "{[\hat{-}, \pm]}"] \arrow[d, "1 + \langle \mathit{frontier}, c \rangle^2"'] & B \arrow[d, "c"] \\
M + (M^* \times \mathrm{N})^2 \arrow[r, "l"'] & \mathrm{N}
\end{tikzcd}$$

commutes, where l is the coproduct $[0, \oplus]$ and the binary operator $\oplus$ is given by

$$(x_0, a) \oplus (x_1, b) \;=\; a + b + w(x_0, x_1) \; .$$

This property suffices to derive an algorithm for k, and the relevant theorem is very similar to our earlier result about dynamic programming. There are however two significant differences that ought to mentioned before stating the theorem.

The first difference is that c is not reqnired to be a fold — we already discussed this for the bracketing problem. Instead, c should satisfy the eqnation

$$c \circ \mu(F) \;=\; l \circ F(\langle\!|g|\!\rangle, c)$$

for some suitable choice of l. In this sense the new theorem is a generalisation of our earlier result, because if $c = \langle\!|h|\!\rangle$, we can take $l = h \circ F(\pi_2)$.

The second difference between the new theorem and the earlier result concerns the F^*-algebra g. So far, g was allowed to be an arbitrary relation, and the loading example showed that this generality is nseful. In the theorem below, however, g is restricted to be a partial arrow. In this sense the new theorem is less general than the earlier result. Why is the restriction to a partial arrow necessary? The proof of the theorem below makes use of the equation

$$(S,1)^\star \circ S^\circ \;=\; (1,S^\circ)^\star\ .$$

This is a valid identity if and only if S is a partial arrow. In the theorem below, it is applied with $S = \llparenthesis g \rrparenthesis$, which is a partial arrow if g is one.

Theorem *Let $\mathcal{E}$ be a Boolean topos with a natural numbers object. Let F be a polynomial endofunctor on $\mathcal{E}$, and let $g : F(B) \to B$ be an $F^\star$–algebra that is a partial arrow in $\mathcal{E}$. Let c and l be arrows of $\mathcal{E}$ that make*

$$\begin{tikzcd}
F(T) \arrow[r, "F^\star(\llparenthesis g \rrparenthesis{,}\, c)^\star"] \arrow[d, "\mu(F)"'] & F(B \times A) \arrow[d, "l"] \\
T \arrow[r, "c"'] & A
\end{tikzcd}$$

commute. Finally, let R be a well–founded preorder on A and define

$$k \;=\; (\,min(R) \circ \exists(c) \circ Inv(\llparenthesis g \rrparenthesis)\,)^{\smile}\ .$$

If l is monotonic with respect to R,

$$\begin{tikzcd}
A \arrow[d, "R"'] & F(B \times A) \arrow[l, "l"'] \arrow[d, "(F \circ (B \times \text{-}))^\star(R)"] \arrow[dl, phantom, "\boxed{?}"] \\
A \arrow[r, "l^\circ"'] & F(B \times A)
\end{tikzcd}$$

then the following diagram (semi–)commutes

$$\begin{tikzcd}
B \arrow[r, "Inv(g)"] \arrow[d, "k"'] & PF(B) \arrow[d, "\exists^\star(t)"] \arrow[dl, phantom, "\boxed{?}"] \\
\tilde{A} & P(\tilde{A}) \arrow[l, "sel(R)"]
\end{tikzcd}$$

where t is given by

$$\begin{tikzcd}
F(B) \arrow[r, "F^{\star}\langle 1, k\rangle^{\star}"] \arrow[d, "t"'] & F(B \times \tilde{A}) \arrow[d, "\overline{(F \circ (B \times -))}(A)"] \\
\tilde{A} & (F(B \times A))^{\sim} \arrow[l, "E(l)"]
\end{tikzcd}$$

3 Discussion

3.1 Related Work

Tbe desire to give a satisfactory treatment of dynamic programming was the original incentive for the work reported here. Of course, we are not the first to give a rigorous formulation of dynamic programming, and the present approach has been much influenced by earlier work of others. In this section we sball attempt to put the results of this thesis into a historical perspective.

In 1957, Bellman published a book entitled *Dynamic Programming* [9]. It describes methods for reasoning about processes where a sequence of decisions has to be taken. Typical applications include inventory control, equipment replacement and other problems where time plays a significant rôle. Tbe adjective *dynamic* in *dynamic programming* is meant to emphasise that time dependence. Perhaps unexpectedly, the noun *programming* does not refer to computer programming. In the present context, it would have been more appropriate to speak of *dynamic planning* — planning a sequence of decisions, one at each point in time. Bellman did not give a precise definition of dynamic programming, and merely used the term as a collective name for the mathematical tools presented in his book. Despite the efforts to formalise dynamic programming, recent textbooks still take this view [32].

During the 33 years that passed since the publication of his introductory book, Bellman's work has found hundreds of practical applications. Among these applications are many examples where time does not enter the picture; the typical sequential nature of dynamic programming has become less important over the years. This is especially true of computing science applications that involve some tree type other than lists. Due to the lack of communication between operations researchers and computing scientists, dynamic programming has developed further in both fields separately. A typical example of this disparity is the common opinion among computing scientists that dynamic programming is a bottom–up tabulation tecbnique for recursive program schemes *e.g.* [1, 69]. An operations researcher views tabulation as a particular way of evaluating the recursion scheme that has been derived by means of dynamic programming.

In his 1957 book, Bellman had already noticed that the use of dynamic programming is governed by the *principle of optimality*. Roughly, tbe principle says tbat an optimal solution to a complex prohlem is composed of optimal solutions to subproblems. Bellman's original formulation of the principle is too vague for use in rigorous program development, but formalising Bellman's work, several authors have given definitions of the principle of optimality, thus rendering dynamic programming a theorem rather than a heuristic.

The first efforts directed towards a formal model of dynamic programming were based on automata theory. In 1967, Karp and Held [58] defined the notion of *discrete decision process* as a means for expressing optimisation problems. They showed how under a certain monotonicity condition, a discrete decision process can be expressed as a *sequential decision process*, which models the concept of a dynamic programming algorithm.

In our approach, the work of Karp and Held can be understood as dynamic programming with a specific data type, namely finite lists. Their notion of *discrete decision process* consists of a function $t : A \rightarrow P(B^*)$ which returns a set of sequences, and a function $f : B^* \rightarrow \mathrm{R}$ which yields the cost of a given sequence. The optimisation problem that is to be solved is

$$k \;=\; min(f^\circ \circ (\leq) \circ f) \;\circ\; t\;.$$

Our generic specification corresponds to Karp and Held's *sequential decision process*. A *monotone* sequential decision process is an instance of the generic specification which satisfies the applicability conditions of the theorem presented in section 2.3. Indeed, that theorem is a generalisation of theorem 1 in Karp and Held's paper [58]. The main concern of that paper is to determine when a discrete decision process can be expressed as a sequential decision process. In this thesis, we have not attempted to address this question. The reason for not doing so is pragmatic: our generic specification is very simple, and it is not difficult to cast a problem into that form. This is not the case for Karp and Held's definition of sequential decision processes, which is obscured by the encoding in the language of automata theory.

Karp and Held's work was generalised by Helman and Rosenthal [50, 51] who proposed a new model of dynamic programming in 1985. They motivated their approach by the observation that certain algorithms cannot be explained in Karp and Held's model: a typical example is the bracketing problem discussed in section 2.3. In terms of the work presented here, Helman and Rosenthal generalised from dynamic programming with lists to more general tree types. Helman and Rosenthal do not explicitly introduce the notion of data types, however: they encode all instances of the generic specification as bracketing problems.

In Helman and Rosenthal's approach, the specification is phrased as a *discrete optimisation problem*. A discrete optimisation problem consists of two components: a *problem structure* and a *choice* function. The choice function corresponds to our selector, and tbe problem structure is another way of describing the composite

$$\exists\,(\!|h|\!) \circ \mathit{Inv}(\!|g|\!)\ .$$

As mentioned above, Helman and Rosenthal only consider one kind of functor, namely $F(B) = A + (B \times B)$, for some given set A of *atoms*. The F-algehras g and h are given as congruence relations on the initial algebra,

$$(\sim) = (\!|g|\!)^\circ \circ (\!|g|\!) \quad \text{and} \quad (\cong) = (\!|h|\!)^\circ \circ (\!|h|\!)\ .$$

The input (the argument to the generic specification) is specified by means of a *non–associative regular expression* that singles out a specific congruence class in $\sim$. At the same time, it specifies a way of enumerating the members of that congruence class. If the principle of optimality is satisfied, the selector can be pushed into this enumeration scheme. The principle of optimality is formalised by a distributivity condition that is similar to ours.

Our approach is a natural extension of Helman and Rosenthal's pioneering efforts, and it seems worthwhile to point out in what sense it is an improvement. The theory presented here takes advantage of recent trends in computing, notably the strong dependence on initial data types, and the use of a calculus of relations. In this sense, it breaks down the artificial barrier between algebraic approaches to algorithm design (as pursued by operations researchers) and the work on formal program development. Apart from this conceptual improvement, the high level of abstraction has the advantage of conciseness and almost mechanical proofs.

The issue of mechanisation brings us to the work of Smith, whose research objectives are closely related to those of this thesis. His goal is to axiomatise classes of algorithms, and to use these axioms in the (semi–)mechanical derivation of computer programs. In his earlier work he concentrated on *divide–and–conquer* and *global search* algorithms, but more recently he has also considered dynamic programming under the name *problem reduction generators*. To discuss problem reduction generators, we first need a summary of Smith's theoretical framework.

A *theory* is a many–sorted theory in classical, first–order predicate logic. Concrete programming problems are modelled by so–called *problem theories*. A problem theory describes the operators and data types for the problem at hand, and for each problem theory there is a single, specified model. It is not quite clear from the paper by Smith and Lowry [90] how this particular model is obtained; becanse of the powerful logic used, initial models cannot be taken for granted. Classes of algorithms are modelled by *algorithm theories*. In contrast to problem theories, no particular model is specified. Algorithm theories are supposed to be given in advance (by an expert), and need not be formulated in the design process of a particular program.

If one wants to develop, say, a divide–and–conquer algorithm for sorting the procedure is as follows. First, introduce a problem theory which specifies an element type, an ordering, bags and lists, together with a predicate that tells whether a list is a sorted bag. Typically, the intended semantics of a problem theory is an initial algebra. To design a divide–and–conquer algorithm, one starts by selecting the relevant algorithm theory. The proof obligation is then to show that the problem theory of sorting can be viewed as an instance of the algorithm theory of divide–and–conquer. This notion of a *view* from an algorithm theory to a problem theory has been borrowed from algebraic specification languages, *e.g.* OBJ3 [45]. Smith has built a computer system that supports the verification of views [88]. Not surprisingly, these verifications are seldom fully mechanised, and the user needs to input monotonicity and distributivity conditions. After it has been shown that the chosen view is correct, a separate component of Smith's computer system synthesises an optimised computer program. Smith has demonstrated the viability of his approach by an impressive number of examples [85, 86, 87, 88, 89, 90].

Smith developed the algorithm theory of *problem reduction generators* [89] while on sabbatical at Oxford, and during that time we often compared notes. It is not surprising, therefore, that our work is quite similar. Problem reduction generators have the same components as the generic specification: there is a signature Σ which plays the rôle of the functor F, there is a Σ–*decomposition structure* which corresponds to the F–algebra g, and a Σ–*composition structure* which corresponds to the F–algebra h. Finally, there is a preorder, just as in the construction of selectors. The main difference is the predicative style enforced by Smith's theorem prover, in contrast to the compositional style enforced by category theory. Smith's work is distinguished from the rest of the algorithm design literature in that it combines abstract theory with mechanisation. I believe that my work is a natural step towards further abstraction, offering the prospect of further mechanisation.

3.2 Future Research

The main shortcoming of this thesis is that it does not show how the results can be used to develop concrete computer programs. This was a conscious decision: the same recursion equation underlies different computer programs on different target architectures, and therefore deriving the equation and implementiug it are separate concerns. There is a large body of literature about implementing the recursion equations that are derived by dynamic programming, *e.g.* [2, 63, 76, 79]. An exciting area for further research is to try and extend our categorical approach to incorporate this aspect of algorithm design as well. Partly speculating, and partly drawing on research that is currently in progress, we shall attempt to outline how this goal migbt be achieved.

The first step is to study the computation of typical inverse images like

$$Inv[1, +\!\!+] \ , \quad Inv[1, +] \ , \quad \text{and} \quad Inv[[\text{-}], +\!\!+] \ ,$$

which perform the splitting of the argument in many dynamic programming algorithms. Preferably, these computations should be expressed as non-recursive equations, using only fold operators as primitives. The use of fold makes it easy to prove properties by initiality, thus avoiding the use of more complicated forms of induction. Once we have a recursion-free characterisation of the splitting function $Inv(g)$, we can substitute it into the right-hand side of the dynamic programming equation

$$k \ = \ m \circ \exists^{*}(h \circ F^{*}(k)) \circ Inv(g) \ ,$$

and then 'promote' $m \circ \exists^{*}(h \circ F^{*}(k))$ into the expression for $Inv(g)$. If the development is successful, this results in a non-recursive expression for k, which is expressed only in terms of folds, and other primitives like products and coproducts. It is important that all component relations in this expression are entire, so they can be implemented as total functions.

A non-recursive expression for k may be directly translated into a categorical programming language like *Charity*, which is currently being developed by Cockett and Fukushima [30]. Charity is a functional programming language, built around the notion of initial algebras for polynomial functors in a distributive category. Charity has the remarkable feature that all programs terminate — an nnexpected result when one considers the expressive power of the language. To make the transition from our topos-theoretic framework to a programming language like Charity precise, it is necessary to show that the programming language can be embedded in a Boolean topos. Furthermore, this embedding should preserve products, coproducts, and initial data-types. Unfortunately, the existence of such an embedding is by no means obvious [29]: more research is needed to link our results to the work on categorical programming languages.

It would be wrong, however, to suggest that a program in Charity is the final stage in the design of a dynamic programming algorithm. In many practical examples, it is still possible to further optimise the code, exploiting further algebraic properties of the operators that are involved in the problem statement. A typical example of such a property arises in the text formatting problem. Let f he the function that returns a measnre of the amount of white space in a single line. A good choice for $f(x)$ is the square of the difference of the optimum line length and the actual length of x itself. It is easily seen that f is *concave*, i.e.

$$f(x +\!\!+ y) - f(y) \ \leqslant f(x +\!\!+ y +\!\!+ z) - f(y +\!\!+ z) \ .$$

Exploiting this property, the complexity of the dynamic programming algorithm can be reduced from $O(n^2)$ to $O(n \log n)$ [41, 52]. It is awkward, however, to express this optimisation in a functional language, as it makes use of binary search in a non–decreasing list of values. This demonstrates the need to integrate the present work with the efforts directed towards a calculus of imperative programs.

An obvious candidate for such integration is the so-called *refinement calculus* developed by Back, Morgan and others [75]. This calculus is built around the notion of *predicate transformers*, which are monotonic functions between power objects. To allow a smooth transition from one formalism to another, it would be nice if all higher–order functions that are defined on relations (especially fold) could also be defined on predicate transformers. Although at the time of writing this issue is still unresolved, there is some evidence which suggests that predicate transformers may be endowed with the same type structure as relations. Predicate transformers form an order–enriched category, and relations are to predicate transformers what functions are to relations. To wit, the predicate transformers with a right adjoint are precisely the relations, and every predicate transformer can be factorised as a span of relations. It appears, therefore, that programming with relations can be generalised to predicate transformers, just as programming with total functions can be generalised to relations. Clare Martin has explored this idea [67] in her thesis, concentrating on the extension of functors and natural transformations. It seems but a small step from her results to proving that fold operators can be defined on predicate transformers as well.

Instead of developing a fully optimised program in an imperative language, one might wish to implement the Charity program in hardware, say, as a systolic array. It is difficult for me to speculate how this might be achieved, because hardware design is completely outside my current field of knowledge. The amount of literature on dynamic programming with systolic arrays suggests, however, that it might be a fruitful area for further research [63, 76]. A good starting point for further investigations is the connection between the present results and the work of Luk, Jones and Sheeran [57, 83] which aims to apply a calculus of relations to hardware design. That calculus is already of a fairly categorical nature [84], and the gap between their work and this thesis seems to be quite narrow.

There is also some existing work on implementing dynamic programming recursions as parallel programs in the P-RAM model [79], but it is less clear what would be an appropriate programming calculus for the P-RAM model than it is for procedural programs or hardware. It would probably be au extension of the refinement calculus, but again my knowledge of the subject is not sufficient to warrant a definite statement.

The primary interest of the research that has been outlined above does not lie in the unification of classes of algorithms. Instead, the main challenge is to provide smooth transitions from one level of abstraction to another. The leading theme in crossing boundaries between programming formalisms is to investigate how the type structure of one formalism can be lifted to another. As was argued above, dynamic programming provides a rich class of examples, not only in terms of specifications, but also in terms of non–trivial implementations on a wide variety of target architectures.

Of course, the identification of further classes of algorithms is also an important research goal. An obvious starting point for further investigations is the class of *greedy* algorithms. Many of these problems can be expressed as instances of the generic specification of dynamic programming. It is to be expected, therefore, that as a by-product of the research on generating splittings, one will also be able to classify a number of tbese algorithms. A slightly different approach is outlined in a forthcoming paper by Richard Bird and myself, where greedy algorithms are classified without relying on the results about dynamic programming that are presented in this thesis [17].

3.3 Conclusions

Summarising, we have seen how dynamic programming is conveniently expressed using the calculus of relations offered by topos theory. The treatment is consistent with other attempts to formalise dynamic programming. Unlike this earlier work, however, our results are also compatible with more general approaches to program construction. In particular, we have made explicit the couuection with the work on initial data types and their use in functional programmiug calculi. The observation that the type structure of functional programming can be generalised to relations is, I believe, one of the most important aspects of this research.

There was little need to develop new mathematics; most of the basic facts were already known to category theorists. I found it fascinating that these existing results, which were invented for truly abstract purposes, are applicable to a subject as mundane as dynamic programming. It is worthwhile to note that the decision to use these abstract results was entirely driven by the applications. Indeed, it was only after the discovery of the main results that I realised how many proofs could be replaced by a reference to the literature.

It was a disappointment that I was nnable to find a definition of *relation totaliser* which works for arbitrary toposes. The restriction to Boolean toposes seems too severe — in the non–Boolean topos $Set^{\to}$ there exists an obvious way to make relations entire, but it is not characterised by my definition.

The restriction to Boolean toposes is, however, a natural one in the context of optimisation problems. For consider a topos $\mathcal{E}$ with a natural numbers object. There exists a canonical partial order on the natural numbers, namely

$$(\leq) \;=\; (+) \circ \pi_2{}^{\circ} \;.$$

Brook has shown that this partial order is well-founded if and only if $\mathcal{E}$ is Boolean ([19], p. 156). It follows that there is little point in developing a theory about optimisation problems for arbitrary toposes.

This brings us to the last conclusion: Boolean toposes provide a convenient set of axioms for deriving programs from specifications. They provide a choice of programming styles: functions, relations and predicate transformers, and smooth transitions between these different programming formalisms. One might argue against this that the axioms of a Boolean topos are not nearly as elementary as (for instance) the axioms of the relational calculus. In the second part of this thesis, however, we shall see how the definition of a Boolean topos may be phrased in terms of relations. Furthermore, it will be shown how these axioms encourage simple algebraic proofs.

Part II

Technical Details

4 Introduction

The first part of this thesis was intended for a wide audience that includes category theorists and computing scientists alike. For this reason the empbasis was on applications and intuitive explanations, not on formal proofs. Indeed, we did not show any of these proofs, and yet it was claimed that their mechanical nature is an attractive feature of this work. The second part of this thesis seeks to substantiate that claim. Here the exposition is aimed at computing scientists with little experience in category theory: many proofs are presented in a calculational style that is popular among researchers in program construction [4, 34, 95]. However, to keep the presentation compact, we shall frequently refer to proofs in the literature. Especially the book *Categories, Allegories* by Freyd and Ščedrov [40] contains many results that are relevant to the present discussion. There is also a handwritten technical report by Carboni, Kelly and Wood about *A 2-categorical approach to geometric morphisms* [23].

4.1 Notation

The intention to write for computing scientists and the desire to make liberal reference to the mathematical literature are somewhat contradictory goals when it comes to notation. On the one hand, computing scientists often prefer notations that are designed for purely syntactic proofs. ("Let the symbols do the work!") On the otber hand, mathematicians prefer the streamlined notations that have been carefully crafted by themselves and their predecessors for many centuries. A mathematician's primary concern is an effective shorthand for communicating his results, and this shorthand should therefore allow reference to the work of others without requiring a tedious translation process. Indeed, at the end of his encyclopedic work on notations, Cajori argues that mathematical progress would benefit by greater symbolic uniformity [20, 21].

It was this second argument that made me chose a very conservative, traditional notation for the first part of this thesis: I did not want to deter any readers who are interested only in learning about the applications, merely by chosing an esoteric notation. In chosing a notation for the second part, I have tried to steer a middle course. The notation is very close to that of Freyd and Ščedrov, so it allows easy reference to their book. However, to meet the exigencies of syntactic proofs, composition is denoted by a semi-colon instead of simple juxtaposition: this

operations

$\Box x$	source of x
$x\Box$	target of x
$x\,;y$	composition of x and y

axioms

$x\,;y$ is defined iff $x\Box = \Box y$

$(\Box x)\Box = \Box x$ and $\Box(x\Box) = x\Box$

$\Box x\,;x = x$ and $x\,;x\Box = x$

$\Box(x\,;y) \succ\!\!- \Box x$ and $(x\,;y)\Box \succ\!\!- y\Box$

Figure 4.1: The definition of categories.

makes it easier to push symbols around without type checking the expressions. The notation is introduced in figure 4.1, which preseuts a definition of categories. Note that objects are notationally confnsed with identity morphisms, and composition is written in diagrammatic order. The asymmetric equality sign ($\succ\!\!-$) means that if the left-hand side is defined, so is the right-hand side and they are equal. Because application of a functor to a morpbism is written in the usual way (Fh), it is convenient to write composition of fnnctors in reversed order, and application of functors associates to the right:

$$(F \circ G)h \;=\; F(Gh) \;=\; FGh\;.$$

4.2 Overview

The structure of part II is as follows. Chapter 5 introduces the basic concepts of the categorical calculus of relations. When a category $\mathcal{E}$ is *regular*, one can construct a category of relations *Rel*($\mathcal{E}$). Conversely, every category of relations arises in this way. This theorem is the starting point of Freyd's theory of *allegories*.

Chapter 6 shows how the basic operators of set theory may be defined in terms of relations. The main idea is to take the isomorphism between relations and set-valued functions as fundamental. If such an isomorphism exists in a regular category, that category is called a *topos*. Like regular categories, toposes can be characterised as categories of relations, this time with some additional structure. Freyd has called such categories of relations *power allegories*. Here we use some basic facts about power allegories to develop a calculus of sets for later applications in dynamic programming.

Chapter 7 reports on an attempt to show that every relation in a topos can be made entire in some canonical way. This attempt was unsuccesful, and I only succeeded in establishing the desired result for *Boolean* toposes. A Boolean topos is a topos where every relation has a complement.

Chapter 8 shows how one may define various optimisation operators in the calculus of relations. The main goal is to establish the distributivity properties of these operators. In particular, it is shown how one may construct a *selector* from a given preorder, using the results about representing partial relations as entire relations. Subsequently, we investigate the algebraic laws of selectors that are constructed in this way.

Chapter 9 presents the theorems about dynamic programming. As we already discussed several applications of these theorems in the first part of this thesis, the emphasis is on formal aspects of the theory. It is shown how the abstract theory may be instantiated in an entirely mechanical fashion.

Finally, in chapter 10, we discuss a number of open problems that arose in the preceding chapters. In particular, it is indicated how one might improve upon the results about making relations entire.

5 Regular Categories

In set theory a relation is defined as a subset of a cartesian product. Accordingly, category theorists view relations as subobjects of products. This chapter starts off by reviewing those properties of subobjects that are relevant to the study of relations. Most of these properties depend on certain assumptions about the category under consideration. This collection of assumptions motivates the definition of a *regular* category. Informally speaking, a regular category is a category where it is easy to reason about subobjects.

Given a regular category $\mathcal{E}$, one may construct the category of relations $Rel(\mathcal{E})$. Its objects are the same as the objects of $\mathcal{E}$, and the arrows $A \to B$ are subobjects of the product $A \times B$. There exists an obvious embedding of $\mathcal{E}$ into $Rel(\mathcal{E})$, which is called the *graph functor*. The algebraic properties of the graph functor will be discussed in detail, and we introduce some elementary operators of the relational calculus.

The algebraic properties of the graph functor are helpful in studying the extension of functors. It is shown how any functor $F : \mathcal{D} \to \mathcal{E}$ can be extended to a mapping $F^* : Rel(\mathcal{D}) \to Rel(\mathcal{E})$. The question is then whether F^* preserves the basic operators on relations: graph, reciprocal (converse), composition, intersection. Graph and reciprocal are always preserved by F^*, and we shall state necessary and sufficient conditions for the preservation of composition and intersection. Finally, it is shown that the extension F^* is in some sense unique.

Are regular categories the natural setting for a calculus of relations? The answer is yes, for there exists an equivalence between the category of small regular categories and the category of small categories of relations. To corroborate this claim, one needs define what *categories of relations* are. There exists such a definition, due to Freyd, which is called the theory of *allegories*. An allegory is a category that satisfies certain axioms, similar to those of classical relation algebras. The category of small *unitary tabular* allegories is equivalent to the category of small regular categories. *Unitary* implies the existence of a unit object (a terminator), and *tabular* means that there are sufficient subobjects to obtain the regular structure.

5.1 Subobjects, Images and Covers

5.1.1 Let h and k be arrows with a common target. The arrow h *factors* through k if there exists an arrow l such that

$$\begin{tikzcd}
\bullet \arrow[rr, "l"] \arrow[dr, "h"'] & & \bullet \arrow[dl, "k"] \\
& \bullet &
\end{tikzcd}$$

commutes. We write $h \preceq k$ to indicate that h factors through k. The relation $\preceq$ is a preorder. Reflexivity is obtained by taking for l the identity arrow, and transitivity follows by pasting two adjacent triangles. Note that identities are maximum among arrows with a given target: $h \preceq h\square$.

5.1.2 When k is monic, there exists at most oue arrow l that makes the above triangle commute. Therefore, if two monics factor through each other, they are isomorphic. To wit, when n and m are monic arrows satisfying

$$n \preceq m \quad \text{and} \quad m \preceq n$$

there exists a unique isomorphism i making

$$\begin{tikzcd}
\bullet \arrow[rr, dotted, "i"] \arrow[dr, tail, "n"'] & & \bullet \arrow[dl, tail, "m"] \\
& \bullet &
\end{tikzcd}$$

commute. (Note that monics are depicted with a crossed tail.) We write $n \sim m$ when n and m are isomorphic. The relation $\sim$ is an equivalence on monics, because $\preceq$ is a preorder. The class of all monics that are isomorphic to n is said to be a *subobject* of $n\square$; it will be denoted $[n]$. The preorder $\preceq$ is extended to a partial order on subobjects in the obvious way:

$$[n] \subset [m] \;\; = \;\; n \preceq m \; .$$

5.1.3 A category is said to be *cartesian* if it has all finite limits. Throughout the remainder of this section, we shall be working in a cartesian category $\mathcal{E}$. The existence of pullhacks implies that the class of subobjects of a given object A forms a semi-lattice. To see this, define the *intersection* of two subobjects $[m]$ and $[n]$ as the diagonal $[p\,;m]$ in the pullback square

$$\begin{tikzcd}
\bullet \arrow[r, tail, "p"] \arrow[d, tail, "q"'] & \bullet \arrow[d, tail, "m"] \\
\bullet \arrow[r, tail, "n"'] & \bullet
\end{tikzcd}$$

Note that p is monic because pullbacks preserve monics. We write

$$[m] \cap [n]$$

for the intersection of $[m]$ and $[n]$. It can be shown that $[m] \cap [n]$ is indeed the greatest lower bound of $[m]$ and $[n]$, so intersection is associative, commutative and idempotent.

5.1.4 The construction of subobjects can be extended to a contravariant functor from $\mathcal{E}$ to $\mathcal{SL}$, the category of semi-lattices and semi-lattice homomorphisms:

$$(\text{-})^{\#} : \mathcal{E}^{op} \to \mathcal{SL} .$$

This functor is called the *subobject* functor. The subobject functor sends an object A to the family $A^{\#}$ of all subobjects of A. On morphisms, $(\text{-})^{\#}$ sends an arrow $f : A \to B$ of $\mathcal{E}$ to its *inverse image function*

$$f^{\#} : B^{\#} \to A^{\#}$$

which is defined as follows. Let $[m]$ be a subobject of B. Then $f^{\#}([m])$ is the subobject $[n]$, where n is the pullback of m along f:

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d, tail, "n"'] & \bullet \arrow[d, tail, "m"] \\
\bullet \arrow[r, "f"'] & \bullet
\end{tikzcd}$$

It follows from the universal property of pullbacks that the function $f^{\#}$ is monotonic, and therefore well-defined. Similarly, one shows that $f^{\#}$ preserves intersections. We have $(f\,;g)^{\#} = g^{\#}\,; f^{\#}$ because whenever the inner squares in

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d, tail] & \bullet \arrow[r] \arrow[d, tail] & \bullet \arrow[d, tail] \\
\bullet \arrow[r, "f"'] & \bullet \arrow[r, "g"'] & \bullet
\end{tikzcd}$$

are pullbacks, so is the outer rectangle. (This property of pullbacks is sometimes called the *pasting property*.)

5.1.5 An arrow $f : A \to B$ is said to have a *direct image function* $f(\text{-}) : A^{\#} \to B^{\#}$ if for all subobjects $[m]$ of A and $[n]$ of B,

$$f([m]) \subset [n] \;\;\Leftrightarrow\;\; [m] \subset f^{\#}([n])\ .$$

In words, $f(\text{-})$ is left adjoint to $f^{\#}$. The subobject $f([A])$ is called the *image* of f. Note that the image of f is the smallest subobject $[n]$ such that f factors through n:

$$f([A]) \subset [n] \;\;\Leftrightarrow\;\; [A] \subset f^{\#}([n]) \;\;\Leftrightarrow\;\; f \preceq n\ .$$

The category $\mathcal{E}$ is said to *have images* if every arrow has a direct image function. Henceforth, we shall assume that the category $\mathcal{E}$ has images.

5.1.6 An arrow $c : A \to B$ is called a *cover* if its image coincides with its target: $c([A]) = [B]$. In the literature, covers are sometimes called *extremal epics*. Covers will be pictured with a crossed arrowhead:

$$\begin{tikzcd} \bullet \arrow[r, "|" marking, pos=0.8] & \bullet \end{tikzcd}$$

The class of covers is closed under left-cancellation, and a monic cover is an isomorphism. The existence of equalisers implies that a cover is epic, and the existence of pullbacks implies that covers compose.

5.1.7 Proposition *A subobject $[m]$ is the image of f iff there exists a cover c such that*

$$\begin{tikzcd} \bullet \arrow[rr, "f"] \arrow[dr, "c"', "|" marking, pos=0.8] & & \bullet \\ & \bullet \arrow[ur, "m"', "|" marking, pos=0.15] & \end{tikzcd}$$

commutes. The pair (c, m) is said to be a cover–monic factorisation of f.

5.1.8 Proposition (e.g. Manes and Arbib [3], p. 39) *If the following is a commuting square where the top is a cover and the bottom a monic,*

$$\begin{tikzcd} \bullet \arrow[r, "|" marking, pos=0.8] \arrow[d] & \bullet \arrow[d] \\ \bullet \arrow[r, "|" marking, pos=0.15] & \bullet \end{tikzcd}$$

then there exists precisely one diagonal arrow as depicted below:

$$\begin{tikzcd}
\bullet \arrow[r, two heads] \arrow[d] & \bullet \arrow[d] \arrow[dl, dotted] \\
\bullet \arrow[r, tail] & \bullet
\end{tikzcd}$$

5.1.9 Pullbacks play an essential rôle in the calculus of suhobjects and images. It seems natural, therefore, to require that pullhacks preserve the image structure, namely monics and covers. Pullbacks always preserve monics. To say that pullbacks preserve covers is to say that one can take arbitrary arrows as representatives of subobjects, not just monics. In a sense this is always possible (two arrows are equivalent if they have the same image), but the point is that it can be done without explicitly using images. To make this statement precise, we need the following definition.

Let h and k be arrows with a common target. The arrow h is *covered* by k if there exist an arrow f and a cover c such that

$$\begin{tikzcd}
\bullet \arrow[r, two heads, "c"] \arrow[d, "f"'] & \bullet \arrow[d, "h"] \\
\bullet \arrow[r, "k"'] & \bullet
\end{tikzcd}$$

commutes. We write $h \lessdot k$ if h is covered by k. Note that h is covered hy a monic m iff h factors through m.

5.1.10 Proposition (Carboni, Kelly and Wood [23], p. 72) *The following two statements are equivalent:*

1. *For all* $h : A \to C$ *and* $k : B \to C$

$$h([A]) \subset k([B]) \quad \Leftrightarrow \quad h \lessdot k \ .$$

2. *Pullbacks preserve covers.*

5.1.11 Now assume that in $\mathcal{E}$, pullbacks do preserve covers. By the above proposition, $\lessdot$ is a preorder, and we can say that two arrows are *equivalent* if they are covered by each other:

$$h \sim k \;=\; h \lessdot k \;\text{ and }\; k \lessdot h \;.$$

This new definition of equivalence is consistent with the earlier definition of equivalence (isomorphism) for monics. In this sense, subobjects can be regarded as equivalence classes of arbitrary arrows in the preorder $\lessdot$. To stress the fact that we allow arbitrary arrows as representatives of subobjects, we shall speak of *extended subobjects*. A monic representative of a subobject is said to be a *tabulation*.

5.1.12 This completes the snmmary of elementary facts about subobjects. images and covers. As we mentioned in the introduction to this chapter, a *regular* category is a category where it is easy to manipulate subobjects. Formally, a category is said to be *regular* if

1. it is cartesian,
2. it has images, and
3. pullbacks preserve covers.

For example, the category of sets and total functions is regular. Grillet's paper [47] contains a much more thorongh discussion of regular categories.

5.1.13 There exists another definition of regularity, which is due to Barr [7]. To conclude this introductory section about regular categories, let us briefly discuss that alternative definition. A *regular epic* is an arrow that occurs as a coequaliser. The following proposition, which Carboni and Street attribute to Joyal, relates regular epics to covers. It also states Barr's definition of regularity, and it says that Barr's definition is equivalent to onrs.

Proposition (A. Joyal, see [25])

- *In any category, a regular epic is a cover.*
- *In a regular category, a cover is a regular epic.*
- *A category is regular iff*
  1. *it is cartesian,*
  2. *every level (kernel pair, pullback of identical arrows) has a coequaliser, which is called its* quotient, *and*

3. *pullbacks preserve regular epics.*

The definition of regularity adopted here is based on tbe notion of subobjects; one might say that it is inspired by set theoretic considerations. In contrast, Barr's definition is of a more algebraic flavour: instead of subobjects, it takes the notion of division by an equivalence relation (a level) as fundamental.

5.2 Relations over a Regular Category

We shall now proceed to define the category of relations over a regular category. Relations will be defined as extended subobjects of binary products. Tbis differs marginally from the definition in the literature, wbere ordinary subobjects are used. I decided to state the definition in terms of extended subobjects because tbat simplifies the notion of composition.

5.2.1 An *order–enriched* category is a category witb a partial order on each homset, where composition is monotonic. Functors between order–enriched categories are also required to be monotonic.

5.2.2 Let $\mathcal{E}$ be a regular category. The *category of relations over* $\mathcal{E}$, denoted $Rel(\mathcal{E})$, is an order–enriched category which is defined as follows.

- The objects are the same as the objects of $\mathcal{E}$.
- The morphisms are extended subobjects of binary products. That is, if $\langle h, k\rangle : E \to (A \times B)$ is a morpbism of $\mathcal{E}$, its equivalence class $[\langle h, k\rangle] : A \to B$ is a morphism of $Rel(\mathcal{E})$.
- Composition is defined as follows. Let $[\langle h, k\rangle]$ and $[\langle h', k'\rangle]$ be morphisms of $Rel(\mathcal{E})$, where k and h' have a common target. Then

  $$[\langle h, k\rangle] ; [\langle h', k'\rangle] \;=\; [\langle r ; h, q ; k'\rangle]$$

  where (r, q) is a pullback of (k, h'):

$$\begin{tikzcd}
 & & \bullet \arrow[dl, "r"'] \arrow[dr, "q"] & & \\
 & \bullet \arrow[dl, "h"'] \arrow[dr, "k"] & & \bullet \arrow[dl, "h'"'] \arrow[dr, "k'"] & \\
\bullet & & \bullet & & \bullet
\end{tikzcd}$$

- The arrow $[(A, A)]$ is the identity on A.
- The partial order on the homsets of $\mathit{Rel}(\mathcal{E})$ is inclusion ($\subset$) of subobjects.

That composition is monotonic (and therefore well-defined) follows from the fact that pullbacks preserve covers. That composition is associative follows from the *pasting property* of pullbacks. The pasting property says that if all squares in

$$\begin{tikzcd}
 & & & \bullet \arrow[dl] \arrow[dr] & & & \\
 & & \bullet \arrow[dl] \arrow[dr] & & \bullet \arrow[dl] \arrow[dr] & & \\
 & \bullet \arrow[dl] \arrow[dr] & & \bullet \arrow[dl] \arrow[dr] & & \bullet \arrow[dl] \arrow[dr] & \\
\bullet & & \bullet & & \bullet & & \bullet
\end{tikzcd}$$

are pullbacks, so are the rectangles. One may conclude that $\mathit{Rel}(\mathcal{E})$ is an order-enriched category. The construction given here is isomorphic to that in the literature by proposition 5.1.10.

5.2.3 The *reciprocal* R° of a relation R is defined by exchanging the branches of a representative:

$$[(h, k)]^\circ \;=\; [(k, h)] \;.$$

Some authors speak of *inverse*, *converse*, *opposite*, *transpose* or *reverse* instead of *reciprocal*. Reciprocation is a contravariant, order-preserving endofunctor on $\mathit{Rel}(\mathcal{E})$, which is furthermore an involution:

$$(R^\circ)^\circ = R \;.$$

5.2.4 There exists an embedding of $\mathcal{E}$ into $\mathit{Rel}(\mathcal{E})$, which is called the *graph functor* G. It is defined as follows:

$$Gf \;=\; [(\Box f, f)] \;.$$

The graph functor is faithful in a very strong sense: if $Gh \subset Gk$, then $h = k$.

5.2.5 Intuitively, morphisms of the form Gh are relations that behave like functions in set theory. To make this precise, say that a relation R is *entire* if

$$\Box R \;\subset\; R\,; R^\circ \;.$$

In the literature, entire relations are sometimes called *total* or *everywhere defined*. Using the following proposition, it is immediate that Gh is entire.

Proposition (Freyd and Ščedrov [40], p. 81) *Let R be a relation. The following three statements are equivalent:*

1. *R is entire.*

2. *There exists a representative (h, k) of R with h a cover.*

3. *For all representatives (h, k) of R, h is a cover.*

5.2.6 A relation R is said to be *simple* if

$$R^\circ ; R \subset R\Box .$$

In category theory, simple relations are more commonly known as *partial* arrows, but in this thesis we shall use the terminology of Freyd and Ščedrov. Of course, any graph Gh is a simple relation, and again this is an instance of a more general result, which is stated below. Notice the subtle difference with the preceding proposition: here the universal quantification is restricted to representatives that are monic (tabnlations).

Proposition (Freyd and Ščedrov [40], p. 81) *Let R be a relation. The following three statements are equivalent:*

1. *R is simple.*

2. *There exists a representative (h, k) of R with h monic.*

3. *For all tabulations (h, k) of R, h is monic.*

5.2.7 A relation that is both entire and simple is called a *map*. The preceding two propositions, together with the fact that monic covers are isomorphisms, yield that a relation is a map iff it is of the form Gh. This result can be strengthened. To state the stronger result, we shall need the notion of *adjoint arrows* in an order–enriched category. Adjoint arrows are discussed in the next paragraph; we return to graphs and maps later.

5.2.8 Let C be an order–enriched category. An arrow h in C is called a *left adjoint* if there exists an arrow in the opposite direction, say h^*, such that

$$h^* ; h \subset h\Box \quad \text{and} \quad \Box h \subset h ; h^* .$$

The arrow h^* is said to be a *right adjoint* of h. The conjunction of two inequations may also be stated as an equivalence:

$$S \subset h ; R \;\Leftrightarrow\; h^* ; S \subset R$$

(for all R and S of appropriate type) or as the symmetrical variant:

$$S ; h \subset R \;\Leftrightarrow\; S \subset R ; h^* .$$

From these two equivalences, one may conclude that the right adjoint h^* is uniquely determined by h. Also, composition of left adjoints gives again a left adjoint. The connection with ordinary adjunctions between partial orders is as follows. An arrow h is a left adjoint in the sense of this paragraph iff $(\text{-} \,;\, h)$ is a left adjoint in the usual sense of the word.

5.2.9 Let us now return to the discussion of maps in the category of relations. As you might expect, an arrow in *Rel*($\mathcal{E}$) is a left adjoint iff it is a map. This yields the following proposition:

Proposition (Carboni, Kelly and Wood [23], p. 81) *Let R be an arrow of Rel($\mathcal{E}$). The following three statements are equivalent:*

1. *R is a left adjoint.*
2. *R is a map.*
3. *There exists an h in $\mathcal{E}$ such that $R = Gh$.*

5.2.10 In words, the preceding proposition says that the original category $\mathcal{E}$ may be recovered from *Rel*($\mathcal{E}$) by taking the subcategory of maps. This observation will be important in the sequel, so let us state it as an explicit proposition:

Proposition *The subcategory of maps Map(Rel($\mathcal{E}$)) is isomorphic to $\mathcal{E}$.*

In view of this result, it seems natural to identify arrows of $\mathcal{E}$ and maps of *Rel*($\mathcal{E}$) in our notation; from now on the graph functor will often be left implicit. There will be occasions, however, where it adds to the clarity of our discussion to denote the graph functor explicitly. We shall continue to make a notational distinction between maps and arbitrary relations: arbitrary relations will be denoted by upper case identifiers $(R, S, T, \ldots)$, while lower case identifiers $(f, g, h, \ldots)$ are reserved for maps.

5.2.11 Recall that a relation R is simple iff it has a representative $\langle h,k\rangle$ with h monic (par. 5.2.6). In particular, we have that m is a monic arrow in $\mathcal{E}$ iff $m\,;m^\circ = \Box m$. This observation can be generalised to obtain an alternative characterisation of tabulations:

Proposition (Freyd and Ščedrov [40], p. 200) *Let $R : A \rightarrow B$ be a relation, and let $h : E \rightarrow A$ and $k : E \rightarrow B$ be maps. Then*

$$h^\circ\,;k = R \ \ \text{and} \ \ h;h^\circ \cap k;k^\circ = E$$

iff $\langle h,k\rangle : E \rightarrow (A \times B)$ is a tabulation of R.

5.3 Extension of Functors

Consider the category of small regular categories. What are the morphisms of this category? You might be inclined to say that they are functors which preserve finite limits and images. However, one could also look at it in a different way. Regular categories were introduced to define relations, and therefore functors between regular categories are those that extend uniquely to order–enriched functors between categories of relations. This section is a summary of what is known about such functors and their extension to relations. It is mostly based on a technical report by Carboni, Kelly and Wood [23].

5.3.1 Let $\mathcal{D}$ and $\mathcal{E}$ be regular categories, and F a functor from $\mathcal{D}$ to $\mathcal{E}$. Define a mapping $F^\star$ from *Rel*($\mathcal{D}$) to *Rel*($\mathcal{E}$) by

$$F^\star R \;=\; (Fm)^\circ\,;Fn$$

where $\langle m,n\rangle$ is a tabulation of R. Because we chose a tabulation of R (and not an arbitrary representative), $F^\star$ is monotonic and well–defined. Furthermore, $F^\star$ agrees with F on maps. This property may be expressed by saying that $F^\star$ distributes through the graph functor:

$$F^\star \circ G \;=\; G \circ F\ .$$

In particular, $F^\star$ preserves identities. Finally, $F^\star$ preserves reciprocals:

$$(F^\star R)^\circ \;=\; F^\star(R^\circ)\ .$$

The mapping $F^\star$ does not preserve composition in any obvious sense.

5.3.2 Proposition (Carboni, Kelly and Wood [23], p. 99) *The following four statements are equivalent:*

1. *For all R and S with $R\square = \square S$,*

   $$F^\star(R\,;S) \;\subset\; F^\star R\,;F^\star S\ .$$

2. *F preserves covers.*

3. *F is monotonic with respect to $(\lhd)$.*

4. *$F^\star$ preserves entireness.*

5.3.3 Let us consider a few examples that illustrate the above proposition. A regular category *satisfies the axiom of choice* if every cover has a left–inverse. The category of sets and total functions satisfies the axiom of choice, and therefore every functor $F : Set \to \mathcal{E}$ preserves covers.

As another example of the above proposition, consider the product functor $\times : (\mathcal{E} \times \mathcal{E}) \to \mathcal{E}$, where $\mathcal{E}$ is regular. Let $c : A \to B$ and $c' : A' \to B'$ be covers. To show that $(c \times c')$ is a cover, it suffices to show that $(c \times A')$ and $(B \times c')$ are covers, because covers compose. Write π_1 for the projection $(A \times A') \to A$. The diagram

$$\begin{tikzcd}
A \times A' \arrow[r, "\pi_1"] \arrow[d, "c \times A'"'] & A \arrow[d, "c"] \\
B \times A' \arrow[r, "\pi_1"'] & B
\end{tikzcd}$$

is a pullback; therefore c cover implies $(c \times A')$ cover. By symmetry, $(B \times c')$ is a cover as well.

Not all functors preserve covers. However, because *Set* satisfies the axiom of choice, it is difficult to give an elementary counter–example. For readers who are familiar with topos theory, it may be helpful to know that in a topos covers coincide with epics. Furthermore, a topos satisfies the *internal axiom of choice* iff the exponent functor $(A \Rightarrow (\text{-}))$ preserves epics for all A ([40], p. 179). This illustrates once more how the extension of functors is intimately connected to the axiom of choice.

5.3.4 The preceding proposition stated a necessary and sufficient condition for F^* to be a weak functor in the following sense:

$$F^*(R\,;S) \ \subset\ F^*R\,;F^*S\ .$$

What if the inequation is reversed? Can one also characterise that situation? The answer is yes, but first we need another definition: to say tbat a functor *preserves pullbacks up to image* is to say that whenever

$$\begin{tikzcd}
\bullet \arrow[r, "q"] \arrow[d, "p"'] & \bullet \arrow[d, "k"] & \text{and} & \bullet \arrow[r, "t"] \arrow[d, "s"'] & \bullet \arrow[d, "Fk"] \\
\bullet \arrow[r, "h"'] & \bullet & & \bullet \arrow[r, "Fh"'] & \bullet
\end{tikzcd}$$

are pullback squares, the mediating arrow in

$$\begin{tikzcd}
\bullet \arrow[dr, dotted] \arrow[drr, "Fq"] \arrow[ddr, "Fp"'] & & \\
& \bullet \arrow[r, "t"] \arrow[d, "s"'] & \bullet \arrow[d, "Fk"] \\
& \bullet \arrow[r, "Fh"'] & \bullet
\end{tikzcd}$$

is a cover. Indeed, when this condition is satisfied, (s, t) is the image of (Fp, Fq) by proposition 5.1.7. If F preserves pullbacks up to image, one can infer a number of simple consequences. For example, F preserves monics and therefore F^* preserves simplicity of relations. Also, F preserves covers iff F preserves images.

5.3.5 Proposition (Carboni, Kelly and Wood [23], p. 98) *The following two statements are equivalent:*

- *For all R and S with $R\Box = \Box S$,*

$$F^*(R\,;S) \ \supset\ F^*R\,;F^*S\ .$$

- *F preserves pullbacks up to image.*

5.3.6 Most functors in computing science preserve pullbacks up to image, but there are a few pathological counter-examples. Consider for instance the functor $F : Set \rightarrow Set$ defined by

$$FA \quad = \quad \begin{cases} \emptyset & \text{if} \quad A = \emptyset \\ \{1\} & \text{if} \quad A \neq \emptyset \end{cases}$$

$$F(h : A \rightarrow B) \quad = \quad \begin{cases} \emptyset \rightarrow (FB) & \text{if} \quad A = \emptyset \\ \{1\} & \text{if} \quad A \neq \emptyset \ . \end{cases}$$

To see that F does not preserve pullbacks up to image, let $h : \{1\} \rightarrow \{0,1\}$ be the constant function returning 0 and let $k : \{1\} \rightarrow \{0,1\}$ be the constant function returning 1. Then $h : k^\circ$ is the empty relation $\emptyset : \{1\} \rightarrow \{1\}$, while

$$F^\star h \, ; F^\star k^\circ \quad = \quad Fh \, ; (Fk)^\circ \quad = \quad \{1\} \quad \not\subseteq \quad \emptyset : \{1\} \rightarrow \{1\} \quad = \quad F^\star(h \, ; k^\circ) \, .$$

By the preceding proposition, F does not preserve pullbacks up to image.

5.3.7 A functor F is called a *relator* if F preserves covers and F preserves pullbacks up to image. The terminology is inspired by the next result, which summarises the previous two propositions.

Proposition (Carboni, Kelly and Wood [23], p. 100) *Let $\mathcal{D}$ and $\mathcal{E}$ be regular categories, and $F : \mathcal{D} \rightarrow \mathcal{E}$. Then $F^\star$ is a functor $Rel(\mathcal{D}) \rightarrow Rel(\mathcal{E})$ iff F is a relator.*

5.3.8 What is the intuition behind relators? We shall examine three typical examples by describing the extended functors in terms of conventional, set theoretic relations. The notation

$$a \, (R) \, b$$

is shorthand for $(a, b) \in R$.

1. Consider the list functor $L : Set \rightarrow Set$. On objects, it takes a set A and returns the set of all finite sequences with elements from A. On arrows, (Lf) is the function that applies f to all elements of a sequence:

   $$(Lf)\,[a_1, a_2, \ldots, a_n] \quad = \quad [f a_1, f a_2, \ldots, f a_n] \, .$$

   A conventional characterisation of the extension of L to relations might read as follows:

   $$\begin{array}{l} \quad [a_1, a_2, \ldots, a_n] \ \ (L^\star(R)) \ \ [b_1, b_2, \ldots, b_m] \\ = \\ \quad (n = m) \ \wedge \ (\forall i : 1 \leq i \leq n : a_i \, (R) \, b_i) \, . \end{array}$$

Indeed, this is the usual way of lifting a given relation R to the data type of lists.

2. As a second example, let A be a set, and consider the exponential functor $(A \Rightarrow) : \mathit{Set} \rightarrow \mathit{Set}$. It takes a set B to the set $(A \Rightarrow B)$ of functions from A to B. On arrows, it is defined as follows:

$$A \Rightarrow (f : B \rightarrow C) \;=\; (\lambda k.k\,;f) : (A \Rightarrow B) \rightarrow (A \Rightarrow C)\;.$$

   As expected, $(A \Rightarrow)^\star$ is the usual way of lifting a given relation R to function spaces:

$$f\;((A \Rightarrow)^\star(R))\;g \;=\; \forall\, a \in A : (f\,a)\,(R)\,(g\,a)\;.$$

3. Finally, let $\exists : \mathit{Set} \rightarrow \mathit{Set}$ be the covariant powerset functor that sends a function to its existeutial image. It is defined by the following equations:

$$\begin{aligned} \exists\, A &= \{x \,|\, x \subseteq A\} \\ (\exists\, f)\,x &= \{f\,a \mid a \in x\}\;. \end{aligned}$$

   When extended to relations, it gives rise to the so–called *Egli–Milner ordering*, which is useful in describing the semantics of parallel programs [77]:

$$\begin{aligned} x\,(\exists^\star(R))\,y \;=\; & (\forall\, a \in x : \exists\, b \in y : a\,(R)\,b) \wedge \\ & (\forall\, b \in y : \exists\, a \in x : a\,(R)\,b)\;. \end{aligned}$$

5.3.9 We have seen that for a relator F, $F^\star$ preserves maps, composition and reciprocal. When does $F^\star$ preserve intersection? Intersection of relations can be characterised in terms of products:

$$R \cap S \;=\; \langle A, A\rangle\,;(R \times^\star S)\,;\langle B, B\rangle^\circ\;. \tag{5.1}$$

Therefore, when F is a relator that preserves products, $F^\star$ preserves intersection. This condition is however too strong, and it can be weakened. Recall that tabulations may be described in terms of intersection: a relation R is tabulated by $\langle h, k\rangle$ iff

$$h^\circ\,;k \;=\; R \;\text{ and }\; h;h^\circ \cap k;k^\circ = \Box\langle h, k\rangle\;.$$

Say that a functor F *preserves tabulations (of relations)* if $\langle h, k\rangle$ monic implies $\langle Fh, Fk\rangle$ monic.

Proposition (Freyd [39]) *Let F be a relator. The following three statements are equivalent:*

1. *$F^\star$ preserves intersection.*

2. *F preserves tabulations.*
3. *F preserves pullbacks.*

Proof We aim to prove the sequence of implications (2) ⇒ (3) ⇒ (1) ⇒ (2).

(2) ⇒ (3) Note that (p,q) is a pullback of (h,k) iff (p,q) is a tabulation of $h\,;k^\circ$. Hence, if F preserves tabulations, F preserves pullbacks.

(3) ⇒ (1) Now suppose that F preserves pullbacks. Consider the pullback

$$\begin{tikzcd}
\bullet \arrow[r, "q"] \arrow[d, "p"'] & \bullet \arrow[d, "\langle f,g\rangle"] \\
\bullet \arrow[r, "\langle h,k\rangle"'] & \bullet
\end{tikzcd}$$

which defines the intersection

$$h^\circ;k \cap f^\circ;g \;=\; [p\,;\langle h,k\rangle]\,.$$

Since F preserves pullbacks,

$$\begin{tikzcd}
\bullet \arrow[r, "Fq"] \arrow[d, "Fp"'] & \bullet \arrow[d, "F\langle f,g\rangle"] \\
\bullet \arrow[r, "F\langle h,k\rangle"'] & \bullet
\end{tikzcd}$$

is also a pullback. Furthermore, if τ is the terminator,

$$\begin{tikzcd}
F(A\times B) \arrow[r, "F\pi_2"] \arrow[d, "F\pi_1"'] & FB \arrow[d, "F!_B"] \\
FA \arrow[r, "F!_A"'] & F\tau
\end{tikzcd}$$

is a pullback, and this implies that $d = (F\pi_1, F\pi_2)$ is monic. We may conclude that

$$\begin{tikzcd}
\bullet \arrow[r, "Fq"] \arrow[d, "Fp"'] & \bullet \arrow[d, "\langle Ff, Fg\rangle"] \\
\bullet \arrow[r, "\langle Fh, Fk\rangle"'] & \bullet
\end{tikzcd}$$

is a pullback, since d is monic and

$$\begin{tikzcd}
\bullet \arrow[r, "F\langle h,k\rangle"] \arrow[d, "\langle Fh, Fk\rangle"'] & \bullet \arrow[dl, "d"'] \\
\bullet & \bullet \arrow[l, "\langle Ff, Fg\rangle"] \arrow[u, "F\langle f,g\rangle"']
\end{tikzcd}$$

commutes. It follows that

$$F^\star(h^\circ;k \cap f^\circ;g) \;=\; F^\star(h^\circ\,;k) \cap F^\star(f^\circ\,;g)\;.$$

That is, $F^\star$ preserves intersection.

(1) ⇒ (2) Finally, suppose that $F^\star$ preserves intersection. Then F preserves tabulations by proposition 5.2.11.

5.3.10 Not all relators preserve intersection. A counter-example is the covariant powerset functor $\exists : \mathit{Set} \to \mathit{Set}$ that sends a function to its existential image. Let $B = \{0,1\}$, and consider the projections

$$\pi_1 : (B \times B) \to B \quad\text{and}\quad \pi_2 : (B \times B) \to B\;.$$

The product arrow $\langle \pi_1, \pi_2\rangle$ is monic, but $\langle \exists\,\pi_1, \exists\,\pi_2\rangle$ is not monic:

$$\langle \exists\,\pi_1, \exists\,\pi_2\rangle\,\{(0,0),(1,1)\} \;=\; \langle \exists\,\pi_1, \exists\,\pi_2\rangle\{(0,1),(1,0)\}$$

It follows that $\exists$ does not preserve tabulations, and therefore $\exists^\star$ does not preserve intersection.

5.3.11 This concludes the discussion to what extent $F^\star$ preserves the structure of $\mathit{Rel}(\mathcal{E})$. The next question is whether $F^\star$ is in some sense unique. Say that a functor $\mathit{Rel}(\mathcal{D}) \to \mathit{Rel}(\mathcal{E})$ *extends* $F : \mathcal{D} \to \mathcal{E}$ if it agrees with F on maps:

$$H \circ G \;=\; G \circ F\;.$$

Are there other functors besides F^* that extend F? It seems unlikely, as there exists no obvious alternative for the definition of F^*. The next proposition confirms that intuition.

Proposition (Gardiner [42]) *Let $\mathcal{D}$ and $\mathcal{E}$ be regular categories, and let $F : \mathcal{D} \to \mathcal{E}$, $H : Rel(\mathcal{D}) \to Rel(\mathcal{E})$ be functors. If H extends F, then $H = F^*$.*

Proof Assume that H extends F, and let $(h,k) : E \rightarrowtail (A \times B)$ be monic. It suffices to show that $Hh^\circ = (Fh)^\circ$, for then we have

$$H(h^\circ ; k) \;=\; Hh^\circ ; Hk \;=\; (Fh)^\circ ; Fk \;=\; F^*(h^\circ ; k) \; .$$

By the uniqueness of right adjoint arrows,

$$Hh^\circ ; Fh \;\subset\; FA \quad \text{and} \quad FE \;\subset\; Fh ; Hh^\circ$$

imply $Hh^\circ = (Fh)^\circ$. The first containment is proved below; the proof of the second containment is analogous.

$$\begin{array}{ll}
 & Hh^\circ ; Fh \\
= & \quad \{ \; H \text{ extends } F \; \} \\
 & Hh^\circ ; Hh \\
= & \quad \{ \; H \text{ functor } \; \} \\
 & H(h^\circ ; h) \\
\subset & \quad \{ \; h \text{ map, } H \text{ monotonic } \; \} \\
 & HA \\
= & \quad \{ \; H \text{ extends } F \; \} \\
 & FA
\end{array}$$

5.4 Allegories

Is it possible to characterise those categories which arise as the category of relations over a regular category? The answer is yes: Freyd has found a simple characterisation of categories of relations in terms of three operations: composition, intersection and reciprocation. These operations satisfy a number of axioms, which constitute the logical theory of *allegories*.

operations

R°	reciprocal of R
$R \cap S$	intersection of R and S

axioms

$$\begin{array}{ll}
\Box R^\circ = R\Box & R^\circ\Box = \Box R \\
(\Box R)^\circ = \Box R & (R\,;S)^\circ = S^\circ\,;R^\circ \\
(R^\circ)^\circ = R & \\
 & \\
R \cap R = R & R \cap S = S \cap R \\
R \cap (S \cap T) = (R \cap S) \cap T & R\,;(S \cap T) \subset R;S\ \cap\ R;T \\
\Box(R \cap S) \mathrel{\succ\!\!-} \Box R & \\
 & \\
(R \cap S)^\circ = R^\circ \cap S^\circ & R;S\ \cap\ T \subset (R\ \cap\ T;S^\circ)\,;\,S
\end{array}$$

Figure 5.1: The definition of allegories.

5.4.1 An *allegory* is a category that has the additional structure displayed in figure 5.1. The notation $R \subset S$ is shorthand for $R = R \cap S$. Most of the equations are obvious, but there is one notable exception: the so–called *modular law*

$$R;S\ \cap\ T\ \subset\ (R\ \cap\ T;S^\circ)\,;S\,. \tag{5.2}$$

The modular law is more commonly known as *Dedekind's rule* [80]. It is usually stated as

$$R;S\ \cap\ T\ \subset\ (R\ \cap\ T;S^\circ)\,;(S\ \cap\ R^\circ;T)$$

which, in the presence of the other axioms, is equivalent to the modular law. A detailed discussion of various formulations of the modular law can be found in [6], appendix B. In what follows, we shall often use the following special case of the modular law:

$$R;h\ \cap\ T\ =\ (R\cap\ T;h^\circ)\,;h\,. \tag{5.3}$$

5.4.2 Proposition (Freyd and Ščedrov [40], pp. 79–80) *Let $\mathcal{E}$ be a regular category. Then Rel($\mathcal{E}$) is an allegory.*

5.4.3 Many of the notions that were defined for the special case of *Rel*($\mathcal{E}$) can be generalised to arbitrary allegories. For instance, it makes sense to speak of entire morphisms, *simple* morphisms and *maps*. Also, a pair of maps $h : E \to A, k : E \to B$ is called a *tabulation* of $R : A \to B$ if

$$h^\circ \,;\, k = R \;\text{ and }\; h;h^\circ \cap k;k^\circ = E$$

(*cf.* proposition 5.2.11). The modular law is useful in proving properties of these concepts. For example, it can be used to show that

$$F\,;(R \cap S) = (F;R \;\cap\; F;S)$$

for all simple morphisms F:

$$F;R \cap F;S \;\subset\; F\,;(R \;\cap\; F^\circ;F;S) \;\subset\; F\,;(R \cap S)\,.$$

The reverse containment $F\,;(R \cap S) \subset (F;R \;\cap\; F;S)$ is one of the allegory axioms.

5.4.4 Let C be an allegory. A morphism $R : A \to A$ in C is said to be *coreflexive* if it is contained in A. The family of all coreflexive morphisms on A is denoted by $Cor(A)$. One can think of coreflexive morphisms as subsets of A. Indeed, if C is tabular, coreflexives on A are in one–to–one correspondence with subobjects of A in $Map(C)$; the bijection $A^{\#} \to Cor(A)$ is given by

$$[m] \;\mapsto\; m^\circ\,;m\,.$$

In any allegory, a coreflexive relation is symmetric ($R^\circ = R$) and idempotent ($R\,;R = R$). As an example, let us prove that coreflexive implies symmetric; idempotency is proved in a similar fashion. First note that by the modular law,

$$R \;=\; \Box R;R \cap R \;\subset\; (\Box R \cap R;R^\circ)\,;R \;\subset\; R\,;R^\circ\,;R$$

for any R. Now assume that R is coreflexive:

$$R \;\subset\; R\,;R^\circ\,;R \;\subset\; A\,;R^\circ\,;A \;=\; R^\circ\,.$$

The reverse containment follows by substituting R° for R.

5.4.5 The *domain* of a morphism R is defined as

$${}^{<}R \;=\; \Box R \cap R;R^\circ\,.$$

It is characterised among coreflexive morphisms C by the equivalence

$${}^{<}F \subset C \;\Leftrightarrow\; R \subset C\,;R\,.$$

Note that a morphism is entire iff its domain coincides with its source. The domain of the converse of R is called the *range* of R, and it is denoted by $R^{>}$. The next two paragraphs list some properties of the domain operator that will be useful in the sequel.

5.4.6 Proposition (Freyd and Ŝĉedrov [40], pp. 198–199) *In any allegory, the domain operator has the following properties:*

$$
\begin{aligned}
{<}(R\,;S) &= {<}(R\,;{<}S) && (5.4)\\
h\,;{<}R &= {<}(h\,;R)\,;h && (5.5)\\
{<}R\,;h &\subset h\,;{<}(h^{\circ}\,;R) && (5.6)\\
{<}(R\cap S) &= {<}R\cap(S\,;R^{\circ}) && (5.7)\\
S\cap R &\subset {<}R\,;S && (5.8)
\end{aligned}
$$

5.4.7 Proposition (Backhouse et al. [6], p. 155) *Let $\mathcal{D}$ and $\mathcal{E}$ be regular categories, and let $F:\mathcal{D}\to\mathcal{E}$ be a relator. Then F^{*} preserves domains.*

5.4.8 Not every allegory is of the form $Rel(\mathcal{E})$: one needs to impose two additional properties which (unlike the allegory axioms) cannot be phrased as simple identities.

First, one needs to ensure the existence of a terminator in the subcategory of maps. An elegant way of doing this is the following. Say that an object τ in an allegory is a *partial unit* if the identity $\square\tau$ is the maximum morphism from τ to itself. A partial unit τ is called a *unit* if for every object A there exists an entire morphism $A\to\tau$. The terminator in a regular category $\mathcal{E}$ is a unit in $Rel(\mathcal{E})$. An allegory that has a unit is said to be *unitary*.

Proposition (Freyd and Ŝĉedrov [40], p. 202) *Let $\mathcal{C}$ be a unitary allegory. Then τ is a terminator in $Map(\mathcal{C})$. Furthermore,*

$$ {<}(\text{-}) : \mathcal{C}(A,\tau)\to Cor(A) $$

is an isomorphism of semi-lattices. Its inverse is

$$ (\text{-}\,;!_A) : Cor(A)\to\mathcal{C}(A,\tau) $$

where $!_A$ is the unique map $A\to\tau$.

5.4.9 The second addition to the allegory axioms says that there exist enough subobjects — enough to have images and finite limits. Formally, we require that every morphism has a tabulation. If an allegory satisfies this condition, it is said to be *tabular*.

5.4.10 **Proposition** (Freyd and Ščedrov [40], pp. 201–202) *Let C be a unitary tabular allegory. Then Map(C) is a regular category, and Rel(Map(C)) is isomorphic to C.*

5.4.11 Let *Reg* be the category of small regular categories, where the morphisms are functors that preserve finite limits and images. Furthermore, let *All* be the category of small unitary tabular allegories, where the morphisms are functors that preserve reciprocal, intersection and units. By the propositions about extending fnnctors (5.3.7, 5.3.9, 5.3.11), together with the results that relate regular categories to allegories (5.2.10, 5.4.2, 5.4.10) we obtain the promised theorem:

Theorem (Freyd and Ščedrov [40], p. 204) *The category Reg of small regular categories is equivalent to the category All of small unitary tabular allegories.*

5.4.12 Allegories are not the only way to characterise categories of relations. In particular, there exist other characterisations where intersection and reciprocation are not primitive [26, 22]. At the time of writing, I do not have an adequate understanding of this work, and therefore I cannot judge whether those characterisations conld be helpful in the present context.

6 Toposes

A regular category has, in a sense, all tbe structure needed to reason about subobjects. Regular categories are however still a long way from providing a form of set theory: there are no arrows for the membership relation, subset inclusion, or existential quantification. How could one add these set theoretic concepts to the categorical calculus of relations? The main idea is to take the isomorphism between relations $A \rightarrow B$ and set-valued functions $A \rightarrow PB$ as fundamental. This is the definition of a *topos*: a regular category $\mathcal{E}$ where the graph functor $G : \mathcal{E} \rightarrow \mathit{Rel}(\mathcal{E})$ has a right adjoint. In section 6.1 below, we shall consider the definition of a topos in more detail.

The isomorphism between relations and set-valued functions gives another interpretation for the results about extending functors to relations. Not only can we extend functors between toposes to the corresponding categories of relations, we can also extend these functors to set-valued functions. This observation leads to a different characterisation of relators in terms of so-called *cross-operators*. Cross-operators are a generalisation of the polymorphic cartesian product function.

The insight gained by studying cross-operators is then applied to *relational algebras* in a topos. The main result of this section says that the process of extending functors preserves initial algebras. This theorem was proved in a set theoretical context by Eilenberg and Wright; the introduction of cross-operators makes it possible to reproduce their proof in an arbitrary topos.

After this short discussion of applications, we return to the definition of toposes. In the preceding chapter it was shown how regular categories can be defined in terms of allegories. A similar characterisation is possible for toposes, by adding a few operators and axioms to the definition of allegories.

This chapter is closed by a detailed study of the existential image functor. It is sbown that in any topos $\mathcal{E}$, $\exists : \mathcal{E} \rightarrow \mathcal{E}$ is a relator. Furthermore, we give an alternative characterisation of its extension $\exists^{\star}$, which relates it to the *Egli–Milner* ordering in the semantics of programming languages.

6.1 The Definition of a Topos

6.1.1 A *topos* is a regular category $\mathcal{E}$ where the graph functor $G : \mathcal{E} \rightarrow \mathit{Rel}(\mathcal{E})$ has a rigbt adjoint. fn what follows, we shall consider this adjunction in its equational presentation:

$$(\, G, \exists, \{\text{-}\}, \ni \,) \ : \ \mathcal{E} \rightharpoonup \mathit{Rel}(\mathcal{E})$$

is called the *power* adjunction, where $G : \mathcal{E} \rightarrow \mathit{Rel}(\mathcal{E})$ is the left adjoint, $\exists : \mathit{Rel}(\mathcal{E}) \rightarrow \mathcal{E}$ is the right adjoint, $\{\text{-}\} : \mathcal{E} \rightarrow (\exists \circ G)$ is the unit, and $\ni : (G \circ \exists) \rightarrow \mathit{Rel}(\mathcal{E})$ is the counit. Furthermore, we have the so-called *triangular identities*, as illustrated by the commuting triangles below:

$$\begin{tikzcd}
\exists \arrow[d, "\{\text{-}\}\exists"'] \arrow[dr, "\exists"] & \\
\exists \circ G \circ \exists \arrow[r, "\exists\ni"'] & \exists
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
G \arrow[r, "G\{\text{-}\}"] \arrow[dr, "G"'] & G \circ \exists \circ G \arrow[d, "\ni G"] \\
& G
\end{tikzcd}$$

This definition of a topos is not very economical. The proof that it is equivalent to other, more concise definitions in the literature can he found in [40], p. 158. The object $\exists A$ is called the *power object* of A; it is customary to denote it by PA.

6.1.2 The primary example of a topos is the category of sets and total functions, and it will be instructive to interpret the data given above in this specific instance. The right adjoint $\exists : \mathit{Rel}(\mathcal{E}) \rightarrow \mathcal{E}$ sends a relation to its *existential image*

$$(\exists R)x \ = \ \{\, b \mid \exists a : a \in x \wedge a(R)b \,\} \ .$$

It follows that $\exists \circ G$ is the covariant powerset functor that takes a function to its existential image. The unit of the adjunction is $\{\text{-}\} : \mathcal{E} \rightarrow (\exists \circ G)$, the natural transformation that forms singleton sets:

$$\{\text{-}\}_A a \ = \ \{a\} \ \text{ for } \ a \in A \ .$$

The notation for the counit of the adjnnction,

$$\ni : (G \circ \exists) \rightarrow \mathit{Rel}(\mathcal{E})$$

rightly suggests that this is the collection of *has–element* relations. The union of a family of sets is a natural transformation

$$\bigcup : (\exists \circ G)^2 \rightarrow (\exists \circ G)$$

that may defined as $\bigcup = \exists\ni G$. The triangular identities are simple laws in set theory: a singleton has only one element

$$B \;=\; G\{\text{-}\}_B \,;\, \ni_B$$

and the union of a singleton set gives its element:

$$PB \;=\; \exists\, GB \;=\; \{\text{-}\}_{\exists\, GB} \,;\, \exists\ni_{GB} \;=\; \{\text{-}\}_{PB} \,;\, \bigcup_B \,.$$

This last equation is sometimes called the *one–point rule*. It does not seem to cause confusion if one writes $\exists$ instead of $\exists \circ G$, and from now on we shall do so.

6.1.3 The triple $(\exists, \{\text{-}\}, \bigcup)$ is an example of a *monad*. To explore further properties of the power adjunction, we shall need some basic facts about monads and their relationship with adjunctions. These facts are reviewed below; we return to the power adjunction later. A more gentle introduction to monads that emphasises applications in functional programming is Wadler's paper [96].

6.1.4 Let C be a category. A *monad* in C is a triple

$$(F, \eta, \mu)$$

where $F : C \to C$ is an endofunctor on C, and $\eta : C \to F$ and $\mu : (F \circ F) \to F$ are natural transformations. Furthermore, these operators satisfy the so–called *associative law*

$$\mu F \,;\, \mu \;=\; F\mu \,;\, \mu$$

as well as the *unit laws*

$$F\eta \,;\, \mu \;=\; F \;=\; \eta F \,;\, \mu \;.$$

The calculus induced by the monad laws is rich, and two derived laws that will be expedient in later calculations are the following: for all $h : A \to B$

$$Fh \;=\; F(h \,;\, \eta_B) \,;\, \mu_B \;,$$

and for all $k : A \to FB$

$$k \;=\; \eta_A \,;\, Fk \,;\, \mu_B \;.$$

In some proofs, we shall refer to these properties by the hint : '(F, η, μ) monad'.

As mentioned above, the triple $(\exists, \{\text{-}\}, \bigcup)$ is an example of a monad in *Set*. The monad laws are familiar identities in set theory like the fact that union distributes over itself:

$$\cup\exists \,;\, \cup \;=\; \exists\cup \,;\, \cup \;.$$

6.1.5 Proposition (Huber [53]) *Let* $(F, U, \eta, \epsilon) : \mathcal{C} \rightharpoonup \mathcal{D}$ *be an adjunction. Then*

$$m \;=\; (U \circ F, \eta, U\epsilon F)$$

is a monad in $\mathcal{C}$*. We say that* m *is the monad* defined by *the adjunction* (F, U, η, ϵ).

It follows that in any topos $\mathcal{E}$ the power adjunction defines a monad

$$(\exists, \{\cdot\}, \bigcup) \; .$$

This monad is said to be the *power monad* of $\mathcal{E}$.

6.1.6 Given the fact that an adjunction determines a monad, the next question to ask is wbether every monad can be obtained from an adjunction. This is indeed the case, and there exist two extremal ways of constructing an adjunction that defines the given monad. The following three paragraphs snmmarise one of these constructions, due to H. Kleisli [59].

6.1.7 Let $\mathcal{C}$ be a category, and let $m = (F, \eta, \mu)$ be a monad in $\mathcal{C}$. The *Kleisli category* of m, denoted $\mathcal{C}^m$, is defined as follows. Its objects are the same as those of $\mathcal{C}$. If $h : A \longrightarrow FB$ is an arrow in $\mathcal{C}$, tben

$$h^\flat : A \rightarrow B$$

is an arrow in the Kleisli category $\mathcal{C}^m$. Composition in $\mathcal{C}^m$ is defined by

$$k^\flat \,;\, h^\flat \;=\; (k \,;\, Fh \,;\, \mu_C)^\flat$$

for $k^\flat : A \rightarrow B$, $h^\flat : B \rightarrow C$ in $\mathcal{C}^m$. Finally, the identity arrows in $\mathcal{C}^m$ are given by

$$A \;=\; (\eta_A)^\flat \; .$$

6.1.8 Theorem (Kleisli [59]) *Let* $\mathcal{C}$ *be a category, and let*

$$m \;=\; (F, \eta, \mu)$$

be a monad in $\mathcal{C}$*. Define two functors* $F^m : \mathcal{C} \rightarrow \mathcal{C}^m$ *and* $U^m : \mathcal{C}^m \rightarrow \mathcal{C}$ *by*

$$F^m(f : A \rightarrow B) \;=\; (f \,;\, \eta_B)^\flat : A \rightarrow B$$

and

$$U^m(k^\flat : D \rightarrow E) \;=\; (Fk \,;\, \mu_E) : FD \rightarrow FE \; .$$

Furthermore, define a natural transformation $\epsilon : (F^m \circ U^m) \rightarrow \mathcal{C}^m$ *by* $\epsilon_A = (FA)^\flat$*. Then*

$$(F^m, U^m, \eta, \epsilon) \;:\; \mathcal{C} \rightharpoonup \mathcal{C}^m$$

is an adjunction and this adjunction defines the given monad m.

6.1.9 The adjunction described in the preceding theorem is in a sense initial among the adjunctions that define the given monad. This fact is expressed by the following result, which is known as the *Comparison Theorem for the Kleisli construction.*

Theorem (Schubert [81], p. 330) *Let* $(F, U, \eta, e) : \mathcal{C} \to \mathcal{D}$ *be an adjunction, and let*

$$m = (U \circ F, \eta, U\epsilon F)$$

be the monad it defines in $\mathcal{C}$. *Define a functor* $L : \mathcal{C}^m \to \mathcal{D}$ *by*

$$L(h^\flat : A \to B) \;=\; (Fh\,; e_{FB}) : FA \to FB\ .$$

Then L *is the unique functor such that*

$$U \circ L = U^m \;\text{ and }\; L \circ F^m = F \;\text{ and }\; L(UFA)^\flat = \epsilon_{FA}\ .$$

The functor L *is said to be the* comparison functor. *The comparison functor is an isomorphism iff* F *is bijective on objects.*

6.1.10 Let $\mathcal{E}$ be a topos. The graph functor $G : \mathcal{E} \to \mathit{Rel}(\mathcal{E})$ is a bijection on objects. Therefore, as an immediate consequence of the above theorem, the Kleisli category $\mathcal{E}^p$ of the power monad

$$p \;=\; (\exists, \{\text{-}\}, \bigcup)$$

is isomorphic to the category of relations $\mathit{Rel}(\mathcal{E})$. This means that relations and set-valued functions are not only isomorphic at the level of homsets: they are also isomorphic as categories. The isomorphism is given by the *power transpose* functor $\Lambda : \mathit{Rel}(\mathcal{E}) \to \mathcal{E}^p$

$$\Lambda(R : A \to B) \;=\; (\{\text{-}\}_A\,; \exists R)^\flat\ ,$$

and its inverse

$$\Lambda^{-1}(h^\flat : A \to B) \;=\; h\,; \ni_B$$

which is the comparison functor $\mathcal{E}^p \to \mathit{Rel}(\mathcal{E})$. The Kleisli category $\mathcal{E}^p$ is an order-enriched category, where the partial order on the homsets is inherited from the category of relations $\mathit{Rel}(\mathcal{E})$:

$$(h \subset k) \;=\; (\Lambda^{-1}h \subset \Lambda^{-1}k)$$

for $h, k : A \to B$ in $\mathcal{C}^p$.

6.1.11 The comparison theorem for the Kleisli construction also has other applications that are relevant to computing science. For example, consider the category of simple relations *Simple*$(\mathcal{E})$ over a topos $\mathcal{E}$. It is a theorem of Lawvere and Tierney that the embedding $\mathcal{E} \to$ *Simple*$(\mathcal{E})$ has a right adjoint. The comparison theorem gives the well-known isomorphism between partial functions as simple relations, and their representation by means of pointed sets. The latter representation has heen advocated as a model of exceptions in functional programming languages, *e.g.* [92]. The adjunction between $\mathcal{E}$ and *Simple*$(\mathcal{E})$ will be further discussed in the next chapter.

6.1.12 Let us close tbis section about the definition of toposes by summarising a few fundamental properties that will be useful in the sequel. Proofs of these facts may be found in the book by Freyd and Ščedrov [40], or indeed in any other textbook about topos tbeory.

6.1.13 Proposition *In a topos, covers and epics coincide.*

6.1.14 Proposition *A topos is cocartesian (it has all finite colimits).*

In particular, a topos has an initial object 0. This means that for every pair of objects A and B, there exists a least relation $\emptyset : A \to B$, which is represented by the initial object:

$$\begin{tikzcd}
& 0 \arrow[dl] \arrow[dr] & \\
A & & B
\end{tikzcd}$$

This collection of *zero* relations acts as a zero with respect to composition:

$$R\,;\emptyset \;=\; \emptyset \;=\; \emptyset\,;R\;.$$

The proof relies on the fact that in a topos, any morphism $A \to 0$ is an isomorphism.

6.1.15 Proposition *In a topos, the coproduct injections are monic, and their pullback is the initial object. Furthermore, if*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "k"] \\ C \arrow[r, "h"'] & D \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} A' \arrow[r, "f'"] \arrow[d, "g'"'] & B \arrow[d, "k"] \\ C' \arrow[r, "h'"'] & D \end{tikzcd}$$

are pullback squares, then

$$\begin{tikzcd} A + A' \arrow[r, "{[f, f']}"] \arrow[d, "g + g'"'] & B \arrow[d, "k"] \\ C + C' \arrow[r, "{[h, h']}"'] & D \end{tikzcd}$$

is a pullback square as well.

6.1.16 The above proposition has a number of important consequences for relations. First of all, it implies that the coproduct functor is a relator. Using the extension of the coproduct functor to relations, one can define the least upper hound of two relations in the following way:

$$R \cup S \;=\; [A, A]^{\circ} \, ; (R +^{*} S) \, ; [B, B] \, . \tag{6.1}$$

The least upper bound $R \cup S$ is said to be tbe *union* of R and S. Note that the definition of union is dual to the characterisation of intersections in terms of products (eq. 5.1, par. 5.3.9). It is however not true that all properties of union are dual to those of intersection. For example, composition distributes over union of relations:

$$R \, ; (S \cup T) \;=\; R;S \,\cup\, R;T \, , \tag{6.2}$$

and the domain operator distributes over union as well:

$$^{<}(R \cup S) \;=\; {}^{<}R \cup {}^{<}S \, .$$

Neither of these equations is valid for intersection.

The properties of the coproduct injections which are stated in the above proposition can also he expressed in terms of relations. For example, the fact that ι_1 and ι_2 are monic is expressed by

$$\iota_1 \, ; \iota_1{}^{\circ} = A \quad \text{and} \quad \iota_2 \, ; \iota_2{}^{\circ} = B \, . \tag{6.3}$$

The following equations say that the pullback of the injections is the initial object:

$$\iota_1 ; \iota_2{}^\circ = \emptyset \quad \text{and} \quad \iota_2 ; \iota_1{}^\circ = \emptyset \ . \tag{6.4}$$

Finally, the coproduct injections satisfy a property which is dual to our earlier characterisation of tabulations:

$$\iota_1{}^\circ;\iota_1 \cup \iota_2{}^\circ;\iota_2 \ = \ A+B \tag{6.5}$$

(cf. proposition 5.2.11). From the last three equations, one may deduce that $A+B$ is a coproduct in *Rel*($\mathcal{E}$), where the coproduct arrow is given by

$$[R,S] \ = \ \iota_1{}^\circ;R \cup \iota_2{}^\circ;S \ . \tag{6.6}$$

Because *Rel*($\mathcal{E}$) is isomorphic to its own dual, we conclude that $(A+B)$ is also a product in the category of relations. The paper by Backhouse *et al.* [6] contains an exhaustive treatment of the algebraic identities that may be derived from these observations.

6.2 Cross–operators

In the previous section, it has been shown how the basic operators of set theory can be defined in terms of the power adjunction: membership, singleton, nnion, existential image. Some form of cartesian product (a collection of arrows $\times_{A,B} : PA \times PB \to P(A \times B)$) is still missing. This section is an exploration of various forms of cartesian product in a topos. It will be shown that there exists an isomorphism between such *cross-operators* and extensions of functors to relations.

6.2.1 Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and $F : \mathcal{D} \to \mathcal{E}$ a functor. A *cross-operator* on F is a natural transformation

$$\gamma : (F \circ \exists) \to (\exists \circ F)$$

such that

$$\begin{aligned} F\{\text{-}\} ; \gamma &= \{\text{-}\}F \\ F\bigcup ; \gamma &= \gamma\exists ; \exists\gamma ; \bigcup F \end{aligned}$$

and for all arrows $g, h : A \to PB$ in $\mathcal{D}$

$$g^\flat \subset h^\flat \quad \text{implies} \quad (Fg ; \gamma_B)^\flat \subset (Fh ; \gamma_B)^\flat \ .$$

This last requirement is just a monotonicity property, and the first two eqnations are also quite natural when one considers the types involved:

$$\begin{tikzcd}
F \arrow[d, "F\{\text{-}\}"'] \arrow[dr, "\{\text{-}\}F"] & \\
F \circ \exists \arrow[r, "\gamma"'] & \exists \circ F
\end{tikzcd}
\qquad
\begin{tikzcd}
F \circ \exists \circ \exists \arrow[rr, "F\bigcup"] \arrow[d, "\gamma\exists"'] & & F \circ \exists \arrow[d, "\gamma"] \\
\exists \circ F \circ \exists \arrow[r, "\exists\gamma"'] & \exists \circ \exists \circ F \arrow[r, "\bigcup F"'] & \exists \circ F
\end{tikzcd}$$

6.2.2 True category theorists will recognise a similarity between cross–operators and the notion of *distributive laws*, as introduced by Beck [8]. Indeed, we shall use cross–operators to describe distributivity properties in chapter 8. We are not the first to perceive the importance of Beck's distributive laws in computing science; Poigné has demonstrated how they can be used in the analysis of powerdomains [78]. Another concept that is similar to our cross–operators are the distributive laws introduced by Manes, who also used them in a treatment of nondeterminism [65].

6.2.3 Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and $F : \mathcal{D} \to \mathcal{E}$ a functor. A functor between the corresponding Kleisli categories $K : \mathcal{D}^p \to \mathcal{E}^p$ *extends* F if

$$K \circ F^p = F^p \circ F \; .$$

Because $\Lambda^{-1} \circ F^p = G$, this is consistent with the earlier definition of *extends* in section 5.3.11: $K : \mathcal{D}^p \to \mathcal{E}^p$ extends F if and only if $(\Lambda^{-1} \circ K \circ \Lambda) : \mathit{Rel}(\mathcal{D}) \to \mathit{Rel}(\mathcal{E})$ extends F.

6.2.4 Theorem *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and $F : \mathcal{D} \to \mathcal{E}$ a functor. The following three statements are equivalent:*

1. *F is a relator.*

2. *There exists a cross–operator on F.*

3. *There exists a unique cross–operator on F, namely*

$$(F\dagger_A)^\flat \;=\; \Lambda F^* \Lambda^{-1} (PA)^\flat \; .$$

Proof In a sequence of three lemmas, it will be shown that there exists a bijection between functors that extend F and cross–operators on F; the theorem then follows by propositions 5.3.7 and 5.3.11. Since the proofs are somewhat laborious, the reader may wish to skip them on first reading.

6.2.4.1 Lemma *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and let $F : \mathcal{D} \to \mathcal{E}$ and $K : \mathcal{D}^p \to \mathcal{E}^p$ be functors. Define γ_B by*

$$(\gamma_B)^\flat \;=\; K(PB)^\flat \,.$$

Suppose that K extends F. Then for all $h^\flat : A \to B$ in $\mathcal{D}^p$,

$$(Fh\,;\gamma_B)^\flat \;=\; Kh^\flat \,.$$

Proof

$$
\begin{array}{cl}
 & (Fh\,;\gamma_B)^\flat \\
= & \quad \{\ p = (\exists, \{\text{-}\}, \bigcup) \text{ monad }\ \} \\
 & (Fh\,; \{\text{-}\}_{FPB}\,; \exists\,\gamma_B\,; \bigcup_{FB})^\flat \\
= & \quad \{\ \text{Kleisli construction (par. 6.1.7)}\ \} \\
 & F^pFh\,; \gamma_B{}^\flat \\
= & \quad \{\ K \text{ extends } F\ \} \\
 & KF^ph\,; \gamma_B{}^\flat \\
= & \quad \{\ \text{def. } \gamma\ \} \\
 & KF^ph\,; K(PB)^\flat \\
= & \quad \{\ K \text{ functor }\ \} \\
 & K(F^ph\,; (PB)^\flat) \\
= & \quad \{\ \text{theorem 6.1.8, triangular identity}\ \} \\
 & Kh^\flat
\end{array}
$$

6.2.4.2 Lemma *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and let $F : \mathcal{D} \to \mathcal{E}$ and $K : \mathcal{D}^p \to \mathcal{E}^p$ be functors. Suppose that K extends F. Then γ_A defined by*

$$(\gamma_A)^\flat \;=\; K(PA)^\flat$$

is a cross–operator on F.

Proof We shall check that γ satisfies the four axioms of a cross–operator on F, i.e.

1. $\gamma : (F \circ \exists) \to (\exists \circ F)$
2. $F\{\text{-}\} ; \gamma \;=\; \{\text{-}\}F$
3. $F\bigcup ; \gamma \;=\; \gamma\exists ; \exists\gamma ; \bigcup F$
4. $g \subset h$ implies $Fg ; \gamma_B \subset Fh ; \gamma_B$, for all $g, h : A \to PB$ in $\mathcal{D}$.

We deal with each of these proof obligations in turn.

1. We aim to show that γ is a natural transformation from $F \circ \exists$ to $\exists \circ F$. Let $h : A \to B$ be an arrow in $\mathcal{D}$. By a calculation that is analogous to the proof of lemma 6.2.4.1, one finds that
$$(\gamma_A ; \exists Fh)^\flat \;=\; K(\exists h)^\flat \;.$$
By lemma 6.2.4.1 itself, we have $K(\exists h)^\flat = (F\exists h ; \gamma_B)^\flat$, and therefore
$$F\exists h ; \gamma_B \;=\; \gamma_A ; \exists Fh \;,$$
as required.

2. We aim to show
$$F\{\text{-}\} ; \gamma \;=\; \{\text{-}\}F \;.$$
Let A be an object of $\mathcal{D}$. One may calculate as follows:
$$\begin{array}{ll} & (F\{\text{-}\}_A ; \gamma_A)^\flat \\ = & \quad \{\ K \text{ extends } F\text{, lemma 6.2.4.1 }\} \\ & K(\{\text{-}\}_A)^\flat \\ = & \quad \{\ \text{def. identity in } \mathcal{D}^p\ \} \\ & KA \\ = & \quad \{\ K \text{ extends } F\ \} \\ & FA \\ = & \quad \{\ \text{def. identity in } \mathcal{E}^p\ \} \\ & (\{\text{-}\}_{FA})^\flat \end{array}$$

3. We aim to show:
$$F\bigcup ; \gamma \;=\; \gamma\exists ; \exists\gamma ; \bigcup F \;.$$

Let A be an object of $\mathcal{D}$. Then

$$\begin{aligned}
& (F\bigcup_A \, ; \gamma_A)^\flat \\
= \quad & \{ \ K \text{ extends } F\text{, lemma 6.2.4.1} \ \} \\
& K(\bigcup_A)^\flat \\
= \quad & \{ \ \text{identity arrows} \ \} \\
& K(PPA \, ; \exists PA \, ; \bigcup_A)^\flat \\
= \quad & \{ \ \text{Kleisli construction (par. 6.1.7)} \ \} \\
& K((PPA)^\flat \, ; (PA)^\flat) \\
= \quad & \{ \ K \text{ functor} \ \} \\
& K(PPA)^\flat \, ; K(PA)^\flat \\
= \quad & \{ \ \text{def. } \gamma \ \} \\
& (\gamma_{PA})^\flat \, ; (\gamma_A)^\flat \\
= \quad & \{ \ \text{Kleisli construction} \ \} \\
& (\gamma_{PA} \, ; \exists \gamma_A \, ; \bigcup_{FA})^\flat
\end{aligned}$$

4. Let $g, h : A \to PB$ be arrows in $\mathcal{D}$ such that $g^\flat \subset h^\flat$. The aim is to show

$$(Fg \, ; \gamma_B)^\flat \ \subset \ (Fh \, ; \gamma_B)^\flat \ .$$

By lemma 6.2.4.1, this follows immediately from the monotonicity of K.

6.2.4.3 In the preceding lemma, we have shown that there exists a mapping from functors that extend F to cross–operators on F. By lemma 6.2.4.1 this mapping is injective, since it says that one may recover K from the cross–operator $(\gamma_B)^\flat = KB$ and F itself:

$$Kh^\flat \ = \ (Fh \, ; \gamma_B)^\flat \ .$$

Now suppose that γ is an arbitrary cross–operator on F. Does the above construction of K in terms of F and γ still yield a functor that extends F ?

Lemma *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and $F : \mathcal{D} \to \mathcal{E}$ a functor. Let γ be a cross–operator on F, and define*

$$K : \mathcal{D}^P \to \mathcal{E}^P$$

by

$$\begin{aligned}
KA \ &= \ FA \\
K(h^\flat : A \to B) \ &= \ (Fh \, ; \gamma_B)^\flat : FA \to FB \ .
\end{aligned}$$

Then K extends F and for all A, $(\gamma_A)^\flat = K(PA)^\flat$.

Proof We shall check the following proof obligations in turn:

1. K is a morphism of graphs.
2. K preserves identity arrows.
3. K preserves composition.
4. K is monotonic.
5. $K \circ F^p = F^p \circ F$.
6. $(\gamma_A)^\flat = K(PA)^\flat$.

1. We aim to show that K is a morphism of graphs.

$$\begin{array}{ll} & h^\flat : A \to B \quad \text{in } \mathcal{D}^p \\ \Leftrightarrow & \quad \{ \text{ def. } \mathcal{D}^p \} \\ & h : A \to PB \quad \text{in } \mathcal{D} \\ \Rightarrow & \quad \{ F : \mathcal{D} \to \mathcal{E} \} \\ & Fh : FA \to FPB \quad \text{in } \mathcal{E} \\ \Rightarrow & \quad \{ \gamma : F \circ \exists \to \exists \circ F \} \\ & (Fh\,; \gamma_B) : FA \to PFB \quad \text{in } \mathcal{E} \\ \Leftrightarrow & \quad \{ \text{ def. of } \mathcal{E}^p \} \\ & (Fh\,; \gamma_B)^\flat : FA \to FB \quad \text{in } \mathcal{E}^p \\ \Leftrightarrow & \quad \{ \text{ def. of } K \} \\ & Kh : KA \to KB \quad \text{in } \mathcal{E}^p \end{array}$$

2. We aim to show that K preserves identity arrows.

$$\begin{array}{ll} & K(\{\text{-}\}_A)^\flat \\ = & \quad \{ \text{ def. of } K \} \\ & (F\{\text{-}\}_A\,; \gamma_A)^\flat \end{array}$$

$$
\begin{aligned}
= \quad & \{\ \gamma \text{ cross–operator on } F\ \} \\
& (\{\text{-}\}_{FA})^\flat \\
= \quad & \{\ \text{def. of } K\ \} \\
& (\{\text{-}\}_{KA})^\flat
\end{aligned}
$$

3. We aim to show that K preserves composition. Let $h^\flat : A \to B$ and $g^\flat : B \to C$ be arrows in $\mathcal{D}^p$.

$$
\begin{aligned}
& K(h^\flat \,;\, g^\flat) \\
= \quad & \{\ \text{def. of } K\text{, Kleisli construction}\ \} \\
& (F(h \,;\, \exists\, g \,;\, \bigcup_C) \,;\, \gamma_C)^\flat \\
= \quad & \{\ F \text{ functor}\ \} \\
& (Fh \,;\, F\exists\, g \,;\, F\bigcup_C \,;\, \gamma_C)^\flat \\
= \quad & \{\ \gamma \text{ cross–operator on } F\ \} \\
& (Fh \,;\, F\exists\, g \,;\, \gamma_{FC} \,;\, \exists\, \gamma_C \,;\, \bigcup_{FC})^\flat \\
= \quad & \{\ \gamma : F \circ \exists \to \exists \circ F\ \} \\
& (Fh \,;\, \gamma_{FB} \,;\, \exists\, Fg \,;\, \exists\, \gamma_C \,;\, \bigcup_{FC})^\flat \\
= \quad & \{\ \exists \text{ functor}\ \} \\
& (Fh \,;\, \gamma_{FB} \,;\, \exists\,(Fg \,;\, \gamma_C) \,;\, \bigcup_{FC})^\flat \\
= \quad & \{\ \text{def. } K\text{, Kleisli construction}\ \} \\
& Kh^\flat \,;\, Kg^\flat
\end{aligned}
$$

4. We aim to show that K is monotonic. Let $g^\flat, h^\flat : A \to B$ in $\mathcal{D}^p$ such that $g^\flat \subset h^\flat$. Then

$$
\begin{aligned}
& Kg^\flat \\
= \quad & \{\ \text{def. } K\ \} \\
& (Fg \,;\, \gamma_B)^\flat \\
\subset \quad & \{\ \gamma \text{ cross–operator}\ \} \\
& (Fh \,;\, \gamma_B)^\flat \\
= \quad & \{\ \text{def. } K\ \} \\
& Kh^\flat
\end{aligned}
$$

5. We aim to show that
$$K \circ F^{p} \;=\; F^{p} \circ F\,.$$
Let $h : A \to B$ in $\mathcal{D}$. Then
$$\begin{array}{cl}
 & KF^{p}h \\
= & \quad \{ \text{ def. } F^{p} \text{ } \} \\
 & K(h\,;\{\text{-}\}_B)^{\flat} \\
= & \quad \{ \text{ def. } K \text{ } \} \\
 & (F(h\,;\{\text{-}\}_B)\,;\gamma_B)^{\flat} \\
= & \quad \{ \text{ } F \text{ functor } \} \\
 & (Fh\,;F\{\text{-}\}_B\,;\gamma_B)^{\flat} \\
= & \quad \{ \text{ } \gamma \text{ cross–operator on } F \text{ } \} \\
 & (Fh\,;\{\text{-}\}_{FB})^{\flat} \\
= & \quad \{ \text{ def. } F^{p} \text{ } \} \\
 & F^{p}Fh
\end{array}$$

6. We aim to show that
$$(\gamma_A)^{\flat} \;=\; K(PA)^{\flat}\,,$$
where A is an arbitrary object of $\mathcal{D}$. This is immediate from the definition of K:
$$K(PA)^{\flat} \;=\; (FPA\,;\gamma_A)^{\flat} \;=\; (\gamma_A)^{\flat}\,.$$

This completes the proof of the lemma, and hence of theorem 6.2.4

6.2.5 Cross–operators are ubiquitous; below we give seven examples that frequently occur in computing science.

1. As a very trivial instance, consider the identity functor on a topos $\mathcal{E}$. Its cross–operator is $\exists$, the identity transformation from $\exists$ to itself.

2. Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and A an object of $\mathcal{E}$. The constant functor
$$K_A : \mathcal{D} \to \mathcal{E}$$
maps all arrows in $\mathcal{D}$ to the identity on A. Here the cross–operator is given by
$$(K_A\dagger)_B \;=\; \{\text{-}\}_A\,. \tag{6.7}$$

3. The motivating example for the definition of cross–operators is the product functor
$$\times : (Set \times Set) \to Set \ .$$
Both $(Set \times Set)$ and Set are toposes. The cross–operator $(\times)\dagger$ is the natural transformation that returns the cartesian product of two sets. Naturality means here that
$$(\exists\, f s)\ (\times)\dagger_{C,D}\ (\exists\, g t) \ \ = \ \ \exists\, (f \times g)(s\ (\times)\dagger_{A,B}\ t)$$
where $s \subseteq A$, $t \subseteq B$, $f : A \to C$ and $g : B \to D$. To instantiate the cross–operator axioms, observe that if $\mathcal{E}$ is a topos with power monad $(\exists, \{\text{-}\}, \bigcup)$, $(\mathcal{E} \times \mathcal{E})$ is a topos with power monad
$$(\,(\exists \times \exists), (\{\text{-}\}, \{\text{-}\}), (\bigcup, \bigcup)\,).$$
The first cross–operator axiom therefore reads
$$(\{\text{-}\} \times \{\text{-}\})\,;(\times)\dagger \ \ = \ \ \{\text{-}\}(\times)\ ,$$
and it says that the cartesian product of two singletons is again a singleton. The second cross–operator axiom
$$(\bigcup \times \bigcup)\,;(\times)\dagger \ \ = \ \ (\times)\dagger(\exists \times \exists)\,;\exists(\times)\dagger\,;\bigcup(\times)$$
says that cartesian product distributes over arbitrary unions. The last requirement in the definition of cross–operators states that $(\times)\dagger$ is monotonic with respect to set inclusion.

4. The cross–operator of the coproduct functor
$$+ : (Set \times Set) \to Set$$
is a coproduct itself:
$$(+)\dagger_{A,B} \ \ = \ \ [\exists\, \iota_1, \exists\, \iota_2] \qquad (6.8)$$
where $\iota_1 : A \to (A + B)$ and $\iota_2 : B \to (A + B)$ are the injections into the coproduct.

5. The list functor $L : Set \to Set$ discussed in paragraph 5.3.8 has a cross–operator that is very similar to cartesian product. Informally, it is given by the following set comprehension:
$$L\dagger_A[x_1, x_2, \ldots, x_n] \ \ = \ \ \{\,[a_1, a_2, \ldots, a_n] \mid \forall i : a_i \in x_i\,\}\ .$$

6. Also in paragraph 5.3.8, we considered the exponential functor

$$(A \Rightarrow) : Set \to Set \ .$$

Its cross–operator is

$$(A \Rightarrow)\dagger_C \ (f : A \to PC) \ = \ \{ g : A \to C \mid \forall a \in A : g\,a \in f\,a \} \ .$$

7. Though the above examples may suggest otherwise, it is not always easy to give a succinct description of a cross–operator. An example of this phenomenon is provided by the cross–operator on the existential image functor $\exists$:

$$\begin{array}{l} \quad \exists\dagger_{\mathbb{N}} \{ \{1,2\}, \{4,5\}, \{6\} \} \\ = \\ \quad \{ \ \{1,4,6\}, \{1,5,6\}, \{1,4,5,6\}, \{2,4,6\}, \{2,5,6\}, \{2,4,5,6\} \\ \qquad \{1,2,4,6\}, \{1,2,5,6\}, \{1,2,4,5,6\} \ \} \ . \end{array}$$

This example also shows why the monotonicity condition in the definition of *cross–operator* is necessary: $\gamma = \bigcup ; \{\text{-}\}\exists$ preserves singletous and distributes over union, but it is not monotonic.

6.2.6 The remainder of this section is devoted to developing a small calculus of cross–operators. In this calculus, there is no longer any use for the Kleisli category of the power monad. For this reason, the power transpose ΛR of a relation R will be regarded as an arrow of $\mathcal{E}$, not of the Kleisli category $\mathcal{E}^p$.

To start with, let us mention that $\Lambda^{-1}(PA) = \ni_A$, and that cross–operators may therefore be written in terms of the membership relation:

$$F\dagger \ = \ \Lambda F^\star \ni \ . \tag{6.9}$$

The identity $\Lambda^{-1}(PA) = \ni_A$ is stated in the comparison theorem for the Kleisli construction (par. 6.1.9).

6.2.7 Proposition *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and let $F : \mathcal{D} \to \mathcal{E}$ be a relator. Then for $R : A \to B$ in Rel($\mathcal{D}$)*

$$\Lambda F^\star R \ = \ F\Lambda R \, ; F\dagger_B \ .$$

Proof

$$\Lambda F^\star R$$

$$
\begin{array}{ll}
= & \{\ \Lambda^{-1} \circ \Lambda = 1\ \} \\
 & \Lambda F^{\star} \Lambda^{-1} \Lambda R \\
= & \{\ \Lambda \circ F^{\star} \circ \Lambda^{-1} \text{ extends } F \text{, lemma 6.2.4.1 }\ \} \\
 & F \Lambda R \,;\, F\dagger_B
\end{array}
$$

6.2.8 Proposition *Let $\mathcal{D}$ and $\mathcal{E}$ be toposes, and let $F : \mathcal{D} \longrightarrow \mathcal{E}$ be a relator. Then for $R : A \rightarrow B$ in Rel($\mathcal{D}$)*

$$F \ni R ; F\dagger_B \;=\; F\dagger_A ; \ni F^{\star} R \,.$$

Proof

$$
\begin{array}{ll}
 & F \ni R ; F\dagger_B \\
= & \{\ \text{prop. 6.2.7 }\ \} \\
 & \Lambda F^{\star} \Lambda^{-1} \ni R \\
= & \{\ \text{power adjunction }\ \} \\
 & \Lambda F^{\star} (\ni_A ; R) \\
= & \{\ F^{\star} \text{ functor }\ \} \\
 & \Lambda (F^{\star} \ni_A ; F^{\star} R) \\
= & \{\ \text{power adjunctiou }\ \} \\
 & \Lambda F^{\star} \ni_A ; \ni F^{\star} R \\
= & \{\ \text{cross–operator (eq. 6.9) }\ \} \\
 & F\dagger_A ; \ni F^{\star} R
\end{array}
$$

6.2.9 Proposition *Let $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ be toposes, and let $F : \mathcal{C} \rightarrow \mathcal{D}$ and $H : \mathcal{D} \rightarrow \mathcal{E}$ be relators. Then $H \circ F$ is a relator and*

$$(H \circ F)\dagger \;=\; H(F\dagger) \;;\; (H\dagger)F \,.$$

Proof The fact that $(H \circ F)$ is a relator is trivial. It therefore remains to show that the following diagram commutes:

$$\begin{tikzcd}
H \circ F \circ \exists \arrow[r, "H(F\dagger)"] \arrow[dr, "(H \circ F)\dagger"'] & H \circ \exists \circ F \arrow[d, "(H\dagger)F"] \\
 & \exists \circ H \circ F
\end{tikzcd}$$

Let A be an object of $\mathcal{C}$. We calculate as follows:

$$\begin{array}{ll}
 & (H \circ F)\dagger_A \\
= & \quad \{ \text{ theorem 6.2.4 } \} \\
 & \Lambda(H \circ F)^{*}\Lambda^{-1}(PA) \\
= & \quad \{ \text{ (-)}^{*} \text{ functor } \} \\
 & \Lambda H^{*}F^{*}\Lambda^{-1}(PA) \\
= & \quad \{ \ \Lambda \text{ isomorphism } \} \\
 & \Lambda H^{*}\Lambda^{-1}\Lambda F^{*}\Lambda^{-1}(PA) \\
= & \quad \{ \text{ theorem 6.2.4 } \} \\
 & \Lambda H^{*}\Lambda^{-1}(F\dagger_A) \\
= & \quad \{ \text{ prop. 6.2.7 } \} \\
 & H(F\dagger_A)\ ;\ H\dagger_{FA}
\end{array}$$

6.2.10 Proposition *Let $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ be toposes. Furthermore, let $F : \mathcal{C} \rightarrow \mathcal{D}$ and $H : \mathcal{C} \rightarrow \mathcal{E}$ be relators. Then their product*

$$\langle F, H \rangle : \mathcal{C} \rightarrow (\mathcal{D} \times \mathcal{E})$$

is a relator and for any object A of $\mathcal{C}$:

$$\langle F, H \rangle\dagger_A \;=\; (F\dagger_A, H\dagger_A)\,.$$

The projection functor

$$\Pi_1 : (\mathcal{D} \times \mathcal{E}) \rightarrow \mathcal{D}$$

is also a relator, and

$$(\Pi_1)\dagger \;=\; \exists\Pi_1\,.$$

This proposition may be proved by verifying that the cross–operator axioms hold. As the reader will imagine, such a proof is not particularly enlightening, and therefore it is omitted.

6.3 Relational Algebras

Relational algebras have received considerable attention in the computing literature, notably in connection with nondeterminism. For instance, Eilenberg and Wright [36] show how relational algebras can be used in the study of nondeterministic automata. Extending this work, Goguen and Meseguer [44] use relational algebras in describing the semantics of *recursive parallel nondeterministic flow programs*. Here we recall some of these earlier results, slightly adapted to the present needs.

6.3.1 Let $\mathcal{E}$ be a category and let F be an endofunctor on $\mathcal{E}$. An *F–algebra* is an arrow of type

$$k : FA \to A \, .$$

For example, consider the set L of non–empty finite lists with elements from E. The function $[\text{-}] : E \to L$ takes an element and turns it into a singleton list. The binary operator $(:) : (E \times L) \to L$ takes an element and places it at the front of a list:

$$e_0 : [e_1, e_2, \ldots, e_n] \;=\; [e_0, e_1, e_2, \ldots, e_n] \, .$$

The coproduct of [-] and (:) is of type

$$[[\text{-}],\, (:)] : (E + (E \times L)) \to L \, .$$

It follows that this coproduct is an F–algebra, where $F : \mathit{Set} \to \mathit{Set}$ is the functor defined by

$$\begin{array}{rcl} FA &=& E + (E \times A) \\ Fk &=& E + (E \times k) \, . \end{array}$$

6.3.2 Let $k : FA \to A$ and $l : FB \to B$ be F–algebras. An *F–homomorphism* from k to l is an arrow $h : A \to B$ such that the following square commutes:

$$\begin{tikzcd} A \arrow[r, "h"] & B \\ FA \arrow[u, "k"] \arrow[r, "Fh"'] & FB \arrow[u, "l"'] \end{tikzcd}$$

It may be checked that the composition of two F–homomorphisms is again an F–homomorphism. The F–algebras in $\mathcal{E}$ thus form the objects of a category $\mathcal{E}_F$, where the arrows are F–homomorphisms. For many functors F, this category has an initial object, which we shall denote by

$$\mu(F) \, .$$

If k is another F–algebra, we shall write

$$(\!|k|\!)_F$$

for the unique F–homomorphism from $\mu(F)$ to k. Sometimes, when F is clear from the context and no confusion is possible, we shall drop F from this notation and simply write $(\!|k|\!)$.

6.3.3 An example of an initial F–algebra is the data type of lists discussed in paragraph 6.3.1. In many functional programming languages, one finds the so–called *fold-right* operator on lists. It is usually defined by two recursion equations

$$\begin{array}{rcl} (\mathit{foldr}\,k\,(\otimes))\,[e] & = & k\,e \\ (\mathit{foldr}\,k\,(\otimes))\,(a:x) & = & a \otimes ((\mathit{foldr}\,k\,(\otimes))\,x) \ . \end{array}$$

These equations are equivalent to the statement that $(\mathit{foldr}\,k\,(\otimes))$ is an F-homomorphism:

$$(\mathit{foldr}\,k\,(\otimes)) \ : \ [[\text{-}], (:)] \ \rightarrow \ [k, \otimes] \ .$$

One may prove this equivalence by a simple calculation, using the defining properties of the coproduct. As $[[\text{-}], (:)]$ is the initial F–algebra $\mu(F)$, it follows that $(\mathit{foldr}\,k\,(\otimes)) = (\!|[k,\otimes]|\!)_F$.

6.3.4 Another example of an initial F–algebra is the following definition of natural numbers. Here the functor F is defined as

$$\begin{array}{rcl} FA & = & \tau + A \\ Fh & = & \tau + h \end{array}$$

where τ is the terminator. The initial F–algebra is the coproduct arrow $[0, \mathit{succ}]$. When this initial F–algebra does exist, we say that the category under consideration has a *natural numbers object*.

6.3.5 Homomorphisms on an initial F–algebra occur much more frequently in programming problems than is usually realised. A particularly lucid account of their importance and the relevant properties is Malcolm's recent paper [64]. It would not do justice to Malcolm's work to try to summarise it here; we just quote two propositions that will be useful in later proofs.

Proposition (Malcolm [64]) *Let $\mathcal{E}$ be a category, and let $F : \mathcal{E} \rightarrow \mathcal{E}$ be a functor such that the initial F–algebra exists. Let h and k be F–algebras, and*

$$g : h^{>} \rightarrow k^{>}$$

an arrow in $\mathcal{E}$. If $h\,;g = Fg\,;k$ then

$$(\!|h|\!)_F\,;g \;=\; (\!|k|\!)_F\,.$$

6.3.6 Proposition (Lambek [61]) *Let $\mathcal{E}$ be a category, and let $F : \mathcal{E} \to \mathcal{E}$ be a functor such that the initial F-algebra $\mu(F)$ exists. Then $\mu(F)$ is an isomorphism in $\mathcal{E}$.*

6.3.7 The preceding paragraphs summarised the standard results on F-algebras. We now proceed to consider relational algebras, namely the category

$$\mathit{Rel}(\mathcal{E})_{F^*}\,.$$

In [36], Eilenberg and Wright mentioned in passing that the initial object of this category coincides with $\mu(F)$ — though their definition of algebras is slightly different from ours. The following theorem shows that despite these differences, Eilenberg and Wright's result is still true. Let $\mathcal{E}$ be a topos, and F an endofunctor on $\mathcal{E}$.

Theorem (Eilenberg and Wright [36]) *Let $\mathcal{E}$ be a topos, and let $F : \mathcal{E} \to \mathcal{E}$ be a relator that has an initial F-algebra $\mu(F)$. Then $\mu(F)$ is also an initial F^*-algebra.*

Proof Let $R : FA \to A$ be an F^*-algebra. We have to exhibit a unique arrow $(\!|R|\!)$ that makes

$$\begin{tikzcd}
FT \arrow[r, "\mu(F)"] \arrow[d, "F^*(\!|R|\!)"'] & T \arrow[d, "(\!|R|\!)"] \\
FA \arrow[r, "R"'] & A
\end{tikzcd}$$

commute. Because $\mu(F)$ is an initial F-algebra, $(\!|F\dagger_A\,;\exists\,R|\!)$ is the unique arrow making

$$\begin{tikzcd}
FT \arrow[rr, "\mu(F)"] \arrow[d, "F(\!|F\dagger_A\,;\exists\,R|\!)"'] & & T \arrow[d, "(\!|F\dagger_A\,;\exists\,R|\!)"] \\
FPA \arrow[r, "F\dagger_A"'] & PFA \arrow[r, "\exists\,R"'] & A
\end{tikzcd}$$

commute. Since Λ is an isomorphism and

$$\Lambda^{-1}(\ F(\![F\dagger_A\,;\exists\,R]\!)\ ;\ F\dagger_A\ ;\ \exists\,R\ )\ =\ F^\star(\Lambda^{-1}(\![F\dagger_A\,;\exists\,R]\!))\ ;\ R\ ,$$

it follows that we can take $(\![R]\!) = \Lambda^{-1}(\![F\dagger_A\,;\exists\,R]\!)$.

6.3.8 How could one think of arrows in $\mathit{Rel}(\mathcal{E})_{F^\star}$, and in particular, what is the intuitive interpretation of homomorphisms of the form

$$(\![h]\!)_{F^\star}\ ?$$

Recall the data type of finite lists, as introduced in paragraph 6.3.1. It is the initial F–algebra, where F is the endofunctor ou *Set* given by

$$\begin{array}{rcl} FA & = & E + (E \times A) \\ Fk & = & E + (E \times k)\ , \end{array}$$

for some given set E of elements. The above theorem says that it does not matter whether we consider tbe functional F–algebra $\mu(F)$ or the relational $F^\star$–algebra $\mu(F^\star)$: both denote the same data type. Let us now look at a typical example of a relational homomorphism on that data type.

Define a relation $\square : (E \times E) \rightarrow PE$ by

$$\square\ =\ \pi_1 \cup \pi_2\ .$$

One may think of $\square$ as a binary operator that selects either of its arguments in a nondeterministic fashion. Similarly,

$$(\![[E, \square]]\!)$$

is a nondeterministic mapping that selects an arbitrary element from its argument list. Less operationally speaking, one could say that $(\![[E,\square]]\!)$ is the *has–element* relation on lists.

6.3.9 It is often awkward to check whether a functor is a relator that has an initial algebra. The next few paragraphs present a sufficient criterion to help in that task. It is not a necessary condition, but it covers many examples that arise in computing science.

Let $\mathcal{E}$ be a topos. The class of *polynomial* endofunctors on $\mathcal{E}$ is inductively defined by the following clauses:

1. The identity functor on $\mathcal{E}$ is polynomial.

2. If A is an object of $\mathcal{E}$, the constant functor which maps all arrows to the identity on A is polynomial.

3. If G and H are polynomial functors, then $G\hat{+}H$ and $G\hat{\times}H$ defined by

$$\begin{array}{rcl}(G\hat{+}H)(k) & = & G(k)+H(k)\\ (G\hat{\times}H)(k) & = & G(k)\times H(k)\end{array}$$

   are also polynomial functors.

6.3.10 Proposition *In a topos, polynomial functors are relators that preserve tabulations.*

This proposition is immediate from the fact that in a topos, both the product functor and the coproduct functor preserve epics and pullbacks.

6.3.11 Proposition (Johnstone [56]) *In a topos with a natural numbers object, polynomial functors have initial algebras.*

6.4 Power Allegories

The definition of toposes in terms of regular categories suggests that one can phrase the definition of a topos entirely in terms of relations. What additions should one make to the allegory axioms? One possibility is to take the equational definition of the power adjunction, and phrase it in the calculus of relations. That would not give a set of simple containments, however. It is for example necessary to say that f is a map in

$$f\,;\{\text{-}\}_B \;=\; \{\text{-}\}_A\,;\exists f\;.$$

Hence, a relational formulation of the power adjunction would not be entirely equational.

6.4.1 Freyd has proposed quite a different solution that takes universal quantification as primitive. This contrasts with a relational encoding of the power adjunction, which is based on existential quantification.

The basic operator in Freyd's approach is the *right-quotient* R/S of two relations with a common target. This division operator can be characterised by five

operation

R/S right–quotient of R and S

axioms

$$\begin{array}{rcl} \Box(R/S) & \succ & \Box R \\ (R/S)\Box & \succ & S\Box \\ (R_0 \cap R_1)/S & \subset & R_0/S \,\cap\, R_1/S \\ T & \subset & (T\,;S)/S \\ (R/S)\,;S & \subset & R \end{array}$$

Figure 6.1: The definition of right-division.

equations, which are displayed in figure 6.1. From these equations, one may deduce the following equivalence:

$$T \subset R/S \;\Leftrightarrow\; T\,;S \subset R\,.$$

In words, $((\text{-})/S)$ is right adjoint to $((\text{-})\,;R)$. When you write out the definition of R/S in ordinary set theory, you can see the connection with universal quantification:

$$x(R/S)y \;=\; \forall\, z \;:\; y(S)z \;\Rightarrow\; x(R)z\ .$$

Before we get on to Freyd's relational characterisation of toposes, we explore the algebraic properties of division.

6.4.2 Proposition *In an allegory that has right–division, the following are valid equations:*

$$\begin{array}{rcll}
R/S\,;f^\circ & = & R/(f\,;S) & (6.10) \\
h\,;R/S & = & (h\,;R)/S & (6.11) \\
(R\,;k^\circ)/S & = & R/(S\,;k) & (6.12) \\
(R/S)\,;(S/T) & \subset & R/T & (6.13) \\
R/(S_0\,;S_1) & = & (R/S_1)/S_0 & (6.14) \\
\Box R & \subset & R/R & (6.15) \\
R/R\,;R & = & R & (6.16) \\
R/(R\Box) & = & R & (6.17) \\
R;S \,\cap\, T/S^\circ & = & (R \cap T);S \,\cap\, T/S^\circ & (6.18)
\end{array}$$

operation

$\ni_R$ epsiloff of R

axioms

$$\begin{array}{lcl} \ni_R\Box & = & R\Box \\ \ni_R & = & \ni_{R\Box} \\ \Box R & \subset & (R/\ni_R)\,;(\ni_R/R) \\ \Box\ni_R & \supset & (\ni_R/\ni_R) \cap (\ni_R/\ni_R)^\circ \end{array}$$

Figure 6.2: The definition of power allegories.

6.4.3 Various other division operators can be defined in terms of right–division. An example is *left–division*:

$$S\backslash R \;=\; (R^\circ/S^\circ)^\circ \,.$$

Left–division is entirely symmetrical to right–division and therefore all facts about right–division translate into properties of left–division. For instance, left–division is characterised by the equivalence

$$T \subset S\backslash R \;\Leftrightarrow\; S;T \subset R \,.$$

6.4.4 A *power allegory* is an allegory with right–quotients, plus the additional structure displayed in figure 6.2.[1] Let $\mathcal{C}$ be a power allegory. Then the inclusion of *Map*($\mathcal{C}$) into $\mathcal{C}$ has a right adjoint, analogous to the power adjunction. The basic idea in constructing this adjunction is to define power transpose by

$$\Lambda R \;=\; (R/\ni_R) \cap (\ni_R/R)^\circ \,.$$

The proof that this works can be found in the book by Freyd and Ščedrov [40], p. 236.

6.4.5 Theorem (Freyd and Ščedrov [40], p. 236.) *If $\mathcal{E}$ is a topos, then Rel($\mathcal{E}$) is a unitary tabular power allegory. Conversely, if $\mathcal{C}$ is a unitary tabular power allegory then Map($\mathcal{C}$) is a topos.*

How is this theorem proved? To show that the maps of a unitary tabular power allegory form a topos is easy; we already discussed the relevant construction in the

[1] In their definition of power allegories, Freyd and Ščedrov [40] require the existence of union ($\cup$) and zero ($\emptyset$) in addition to division and epsiloff. On page 250, they prove that the union and zero operators may be omitted.

preceding paragraph. For the other direction, we need to show that every topos has quotients. If we can construct the *containment* relation

$$\sqsupseteq_B = \ \ni_B / \ni_B : PB \rightarrow PB$$

we are done, for then it is possible to define

$$R/S \ = \ \Lambda R \,;\sqsupseteq_B ; (\Lambda S)^\circ \ .$$

Define the *internal intersection map* $\sqcap_A : (PA \times PA) \rightarrow PA$ by

$$\sqcap_A \ = \ \Lambda((\ni_A \times^* \ni_A) \,; \langle A, A\rangle^\circ) \ .$$

Note that for any R and S, $\Lambda(R \cap S) = \langle \Lambda R, \Lambda S\rangle \,; \sqcap_B$. The containment can now be defined as follows:

$$\sqsupseteq_A \ = \ \pi_1{}^\circ \,; (\cap_A \cap \pi_2) \ .$$

The proof that this construction works makes use of the fact that a tahulation of $\pi_1{}^\circ \,; (\sqcap_A \cap \pi_2)$ is an equaliser of $\cap_A$ and π_2.

6.4.6 Note that the above theorem is not stated as an equivalence of categories. Indeed, we did not investigate which relators preserve the additional structure of power allegories. For instance, which functors preserve division? The answer is: hardly any. Even the product functor fails to preserve division, as may be checked by writing out the set theoretical definitions. Still, in the sequel it will sometimes be necessary to distrihute fnnctors over division. For this reason, we shall introduce a variant of left-division, called *strict left-division*, which is preserved by all polynomial functors.

Consider a category that has division, and let R and S be arrows with a common source. The *strict left-quotient* of R and S is defined by

$$R \mathbin{\backslash\!\backslash} S \ = \ R^{>} ; R \backslash S \ . \qquad (6.19)$$

Alternatively, it may be characterised by the equivalence

$$(T \subset R \mathbin{\backslash\!\backslash} S) \ \Leftrightarrow \ (R;T \subset S \ \text{ and } \ {}^{<}T \subset R^{>}) \ . \qquad (6.20)$$

Some authors speak of *weakest postspecification*, *typed factor* or *conjugate kernel* instead of *strict left-division*. Two properties of ($\backslash\!\backslash$) will be useful in later calculations. The first says that ($\backslash\!\backslash$) and $(\text{-})^\circ$ interact like division and reciprocal in number arithmetic:

$$R \mathbin{\backslash\!\backslash} S \subset R^\circ ; S \ . \qquad (6.21)$$

The second property follows from the first, and it says that if m is a monic map,

$$R \mathbin{\backslash\!\backslash} (S \,; m) \ = \ (R \mathbin{\backslash\!\backslash} S) \ ; \ m \ . \qquad (6.22)$$

Note that neither of these identities is valid for ordinary left-division.

6.4.7 Theorem *Polynomial endofunctors on a topos preserve strict left-division.*

Proof It is obvious that the identity functor and constant functors preserve strict left-division. In the next two lemmas, it is proved that the product and coproduct functor preserve strict left-division as well. The theorem then follows by induction.

6.4.7.1 Lemma *In a topos, the product functor preserves strict left-division.*

Proof Let $\mathcal{E}$ be a topos. We aim to show

$$(S_0 \times^\star S_1) \backslash\backslash (R_0 \times^\star R_1) \;=\; (S_0 \backslash\backslash R_0) \times^\star (S_1 \backslash\backslash R_1) \;.$$

The inclusion ($\supset$) is obvious. What about the reverse containment? Observe that the product of $\mathcal{E}$ satisfies the following equations in $Rel(\mathcal{E})$:

$$\begin{aligned} R \times^\star S &= (\pi_1 \,;\, R \,;\, \pi_1{}^\circ) \cap (\pi_2 \,;\, S \,;\, \pi_2{}^\circ) && (6.23) \\ {<}(R \times^\star S) \,;\, \pi_1 \,;\, S &= (R \times^\star S) \,;\, \pi_1 \;. && (6.24) \end{aligned}$$

These identities are discussed in a slightly different setting in [6]; here their proof is omitted. We reason backwards from the proof obligation:

$$\begin{aligned} & (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;\subset\; (S_0\backslash\backslash R_0) \times^\star (S_1\backslash\backslash R_1) \\ \Leftrightarrow \quad & \{ \text{ equation (6.23), symmetry } \} \\ & (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;\subset\; \pi_1 \;;\; S_0\backslash\backslash R_0 \;;\; \pi_1{}^\circ \\ \Leftrightarrow \quad & \{ \; \pi_1 \text{ is a map } \} \\ & \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;;\; \pi_1 \;\subset\; S_0\backslash\backslash R_0 \\ \Leftrightarrow \quad & \{ \text{ strict left-division (eq. 6.20) } \} \\ & S_0 \;;\; \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;;\; \pi_1 \;\subset\; R_0 \quad \text{and} \\ & {<}(\, \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;;\; \pi_1 \,) \;\subset\; S_0{>} \end{aligned}$$

We deal with each of these proof obligations in turn. The first is proved as follows:

$$\begin{aligned} & S_0 \;;\; \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;;\; \pi_1 \\ \subset \quad & \{ \text{ equation (6.23) } \} \\ & S_0 \;;\; \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(\pi_1 \,;\, R_0 \,;\, \pi_1{}^\circ) \;;\; \pi_1 \end{aligned}$$

$$
\begin{array}{ll}
= & \{ \text{ def. } (\backslash\backslash) \text{ (eq. 6.19), division calculus (prop. 6.4.2) } \} \\
& S_0 \;;\; \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(\pi_1 \,;\, R_0) \;;\; \pi_1{}^\circ \;;\; \pi_1 \\
= & \{ \; \pi_1 \text{ epic, proposition 5.2.5 } \} \\
& S_0 \;;\; \pi_1{}^\circ \;;\; (S_0 \times^\star S_1) \backslash\backslash (\pi_1 \,;\, R_0) \\
= & \{ \text{ def. strict left-division } \} \\
& S_0 \;;\; \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)^{>} \;;\; (S_0 \times^\star S_1)\backslash(\pi_1 \,;\, R_0) \\
= & \{ \text{ equation (6.24), def. range } \} \\
& \pi_1{}^\circ \;;\; (S_0 \times^\star S_1) \;;\; (S_0 \times^\star S_1)\backslash(\pi_1 \,;\, R_0) \\
= & \{ \text{ division calculus } \} \\
& \pi_1{}^\circ \;;\; (S_0 \times^\star S_1) \;;\; (\pi_1{}^\circ \,;\, (S_0 \times^\star S_1))\backslash R_0 \\
\subset & \{ \text{ left-divisiou } \} \\
& R_0
\end{array}
$$

It remains to show that

$$
{}^{<}(\, \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)\backslash\backslash(R_0 \times^\star R_1) \;;\; \pi_1 \,) \;\subset\; S_0{}^{>} \;.
$$

Here is a proof:

$$
\begin{array}{ll}
& {}^{<}(\, \pi_1{}^\circ \;;\; (S_0 \times^\star S_1) \backslash\backslash (R_0 \times^\star R_1) \,;\, \pi_1 \,) \\
= & \{ \text{ domain calculus (prop. 5.4.6), } \pi_1 \text{ entire } \} \\
& {}^{<}(\, \pi_1{}^\circ \;;\; (S_0 \times^\star S_1) \backslash\backslash (R_0 \times^\star R_1) \,) \\
\subset & \{ \text{ strict left-division (eq. 6.21) } \} \\
& {}^{<}(\, \pi_1{}^\circ \;;\; (S_0 \times^\star S_1)^\circ \;;\; (R_0 \times^\star R_1) \,) \\
\subset & \{ \text{ domain calculus } \} \\
& {}^{<}(\, \pi_1{}^\circ \,;\, (S_0 \times^\star S_1)^\circ \,) \\
= & \{ \text{ reciprocal, def. range } \} \\
& ((S_0 \times^\star S_1) \,;\, \pi_1)^{>} \\
= & \{ \text{ equation (6.24) } \} \\
& ({}^{<}(S_0 \times^\star S_1) \,;\, \pi_1 \,;\, S_0)^{>} \\
\subset & \{ \text{ domain calculus } \} \\
& S_0{}^{>}
\end{array}
$$

6.4.7.2 Lemma *In a topos, the coproduct functor preserves strict left-division.*

Proof Let $\mathcal{E}$ be a topos. Recall that the coproduct of $\mathcal{E}$ is also a coproduct of $\mathit{Rel}(\mathcal{E})$ (par. 6.1.16). It is therefore not necessary to write $(+^*)$ instead of $(+)$. Furthermore, to say that

$$(S_0 + S_1) \mathbin{\backslash\!\backslash} (R_0 + R_1) \;=\; (S_0 \mathbin{\backslash\!\backslash} R_0) + (S_1 \mathbin{\backslash\!\backslash} R_1)$$

is to say that

$$\iota_1 \,;\, (S_0 + S_1) \mathbin{\backslash\!\backslash} (R_0 + R_1) \;=\; (S_0 \mathbin{\backslash\!\backslash} R_0) \,;\, \iota_1 \;.$$

The latter equation is proved as follows:

$$\begin{aligned}
& \iota_1 \,;\, (S_0 + S_1) \mathbin{\backslash\!\backslash} (R_0 + R_1) \\
= \quad & \{ \text{ def. strict left-division (eq. 6.19) } \} \\
& \iota_1 \,;\, (S_0 + S_1)^{>} \,;\, (S_0 + S_1)\backslash(R_0 + R_1) \\
= \quad & \{ \text{ relators preserve range (prop. 5.4.7) } \} \\
& \iota_1 \,;\, (S_0{}^{>} + S_1{}^{>}) \,;\, (S_0 + S_1)\backslash(R_0 + R_1) \\
= \quad & \{ \; \iota_1 \text{ natural } \} \\
& S_0{}^{>} \,;\, \iota_1 \,;\, (S_0 + S_1)\backslash(R_0 + R_1) \\
= \quad & \{ \text{ division calculus (prop. 6.4.2) } \} \\
& S_0{}^{>} \,;\, ((S_0 + S_1) \,;\, \iota_1{}^{\circ})\backslash(R_0 + R_1) \\
= \quad & \{ \; \iota_1{}^{\circ} \text{ natural } \} \\
& S_0{}^{>} \,;\, (\iota_1{}^{\circ} \,;\, S_0)\backslash(R_0 + R_1) \\
= \quad & \{ \text{ division calculus } \} \\
& S_0{}^{>} \,;\, S_0\backslash(\iota_1 \,;\, (R_0 + R_1)) \\
= \quad & \{ \text{ def. strict left-division } \} \\
& S_0 \mathbin{\backslash\!\backslash} (\iota_1 \,;\, (R_0 + R_1)) \\
= \quad & \{ \; \iota_1 \text{ natural } \} \\
& S_0 \mathbin{\backslash\!\backslash} (R_0 \,;\, \iota_1) \\
= \quad & \{ \; \iota_1 \text{ monic, strict left-division (eq. 6.22) } \} \\
& S_0 \mathbin{\backslash\!\backslash} R_0 \;;\; \iota_1
\end{aligned}$$

6.5 Existential Image

The existential image functor $\exists : \mathcal{E} \to \mathcal{E}$ is a relator. We already mentioned this fact for the special case of sets and total functions, but so far the proof was omitted. That proof is given below; it turns out to be a simple application of the modular law. We then proceed to explore further algebraic properties of $\exists$ in the calculus of relations.

Given the fundamental importance of $\exists$ in the definition of a topos, it is not surprising that its extension also plays a leading rôle. For example, the *containment* relation

$$\sqsupseteq_A \;=\; \ni_A/\ni_A : PA \to PA$$

is a natural transformation from $\exists$ to $\exists^\star$. Using this fact, it can be shown that

$$\exists^\star R \;=\; (\in_A\backslash(R\,;\in_B)) \cap ((\ni_A\,; R)/\ni_B)\ ,$$

which means that $\exists^\star R$ is precisely the *Egli–Milner* ordering from programming language semantics.

6.5.1 Proposition *Let $\mathcal{E}$ be a topos. The existential image functor $\exists : \mathcal{E} \to \mathcal{E}$ is a relator.*

Proof That $\exists$ preserves epics is clear: if e is an epic, then $\exists e$ is a split epic with left-inverse $\exists e^\circ$.

A *weak pullback* is like a pullback, except that the mediating arrow is not required to be unique. Note any two weak pullbacks factor through each other. Therefore, to show that $\exists$ preserves pullbacks up to image, it suffices to show that $\exists : \mathcal{E} \to \mathcal{E}$ preserves weak pullbacks. Since $\exists : \mathit{Rel}(\mathcal{E}) \to \mathcal{E}$ is right adjoint, it preserves weak pullbacks. It remains to show that the graph functor $\mathcal{E} \to \mathit{Rel}(\mathcal{E})$ preserves weak pullbacks. Again, because any two weak pullbacks factor through each other, one only needs to prove that a pullback in $\mathcal{E}$ is a weak pullback in $\mathit{Rel}(\mathcal{E})$. Let the following be a pullback square in $\mathcal{E}$:

$$\begin{tikzcd}
\bullet \arrow[r, "q"] \arrow[d, "p"'] & \bullet \arrow[d, "g"] \\
\bullet \arrow[r, "f"'] & \bullet
\end{tikzcd}$$

Let R and S be relations in $\mathit{Rel}(\mathcal{E})$ such that $R\,; f = S\,; g$. Define

$$T \;=\; R;p^\circ \cap S;q^\circ$$

We aim to show that

$$\begin{tikzcd}
\bullet \arrow[dr, dotted, "T"] \arrow[drr, "S"] \arrow[ddr, "R"'] & & \\
& \bullet \arrow[r, "q"] \arrow[d, "p"'] & \bullet \arrow[d, "g"] \\
& \bullet \arrow[r, "f"'] & \bullet
\end{tikzcd}$$

commutes. As an immediate consequence of tbe modular law, we have that

$$T\,;p \;=\; R \cap S;q^\circ;p\;.$$

Using this equation, we calculate:

$$\begin{array}{ll}
 & R = T\,;p \\
\Leftrightarrow & \quad \{\ \text{above}, \cap \text{ greatest lower bound}\ \} \\
 & R \subset S\,;q^\circ\,;p \\
\Leftrightarrow & \quad \{\ (p,q) \text{ pullback of } (f,g)\ \} \\
 & R \subset S\,;g\,;f^\circ \\
\Leftrightarrow & \quad \{\ f \text{ map}\ \} \\
 & R\,;f \subset S\,;g \\
\Leftrightarrow & \quad \{\ \text{assumption}\ \} \\
 & \text{true}
\end{array}$$

By symmetry, we also have $S = T\,;q$.

6.5.2 Proposition *Let* $T : A \to B$. *If* $T^\circ\,;T = B$, *then*

$$T\,;\in_B \;\subset\; \in_A\,;\exists T\;.$$

Proof

$$T\,;\in_B \;\subset\; \in_A\,;\exists T$$

$$
\begin{array}{ll}
\Leftrightarrow & \{ \text{ reciprocal } \} \\
& \ni_B ; T^\circ \;\subset\; (\exists T)^\circ ; \ni_A \\
\Leftrightarrow & \{ \;\ni : \exists \rightarrow \mathit{Rel}(\mathcal{E}) \; \} \\
& \exists T^\circ ; \ni_A \;\subset\; (\exists T)^\circ ; \ni_A \\
\Leftarrow & \{ \text{ monotonicity } \} \\
& \exists T^\circ \;\subset\; (\exists T)^\circ \\
\Leftrightarrow & \{ \;\exists T \text{ map } \} \\
& \exists T^\circ ; \exists T \;\subset\; PB \\
\Leftrightarrow & \{ \text{ inclusion of maps } \} \\
& \exists T^\circ ; \exists T \;=\; PB \\
\Leftrightarrow & \{ \;\exists \text{ faithful fnnctor } \} \\
& T^\circ ; T \;=\; B
\end{array}
$$

6.5.3 Proposition *Let $R : A \rightarrow B$. If R is entire,*

$$\exists^\star R ; \ni_B \;=\; \ni_A ; R \;.$$

Proof

$$
\begin{array}{ll}
& \exists^\star R ; \ni_B \\
= & \{ \text{ let } \langle e, h \rangle \text{ be a tabulation of } R \; \} \\
& (\exists e)^\circ ; \exists h ; \ni_B \\
= & \{ \;\ni : \exists \rightarrow \mathit{Rel}(\mathcal{E}) \; \} \\
& (\exists e)^\circ ; \ni_C ; h \\
= & \{ \text{ see below } \} \\
& \ni_A ; e^\circ ; h \\
= & \{ \;\langle e, h \rangle \text{ tabulation of } R \; \} \\
& \ni_A ; R
\end{array}
$$

In the penultimate step, it was claimed that

$$(\exists e)^\circ ; \ni_C \;=\; \ni_A ; e^\circ \;.$$

By the properties of reciprocal, this is equivalent to

$$\in_C ; \exists e \;=\; e ; \in_A .$$

We aim to prove the latter identity by mutual inclusion. The containment ($\supset$) is immediate from par. 6.5.2, since R entire implies e cover (par. 5.2.5). The inclusion ($\subset$) follows from the fact that e is entire:

$$\begin{array}{ll}
 & \in_C ; \exists e \;\subset\; e ; \in_A \\
\Leftrightarrow & \quad \{\ \exists e \text{ map }\} \\
 & \in_C \;\subset\; e ; \in_A ; (\exists e)^\circ \\
\Leftrightarrow & \quad \{\ \ni : \exists \dot{\rightarrow} \mathit{Rel}(\mathcal{E}), \text{ reciprocal }\} \\
 & \in_C \;\subset\; e ; e^\circ ; \in_C \\
\Leftrightarrow & \quad \{\ e \text{ entire }\} \\
 & \text{true}
\end{array}$$

6.5.4 Recall the *containment* relation $\sqsupseteq_A : PA \rightarrow PA$

$$\sqsupseteq_A \;=\; \ni_A / \ni_A ,$$

which we briefly discussed in paragraph 6.4.5. From the properties of right–division, it follows that containment is reflexive and transitive:

$$A \subset \sqsupseteq_A \quad \text{and} \quad (\sqsupseteq_A ; \sqsupseteq_A) \subset \sqsupseteq_A .$$

That containment is also anti–symmetric is expressed by one of the power allegory axioms:

$$(\ni_A/\ni_A) \cap (\ni_A/\ni_A)^\circ \subset PA .$$

6.5.5 Theorem *Containment is a natural transformation* $\sqsupseteq : \exists \dot{\rightarrow} \exists^*$.

Proof Below we give the main argument; various details are proved in the lemmas that follow this theorem. Let $\langle h, k\rangle : C \leftarrowtail\!\!\!\longrightarrow (A \times B)$ be a tabulation of $R : A \rightarrow B$.

$$\begin{array}{ll}
 & \exists R ; \sqsupseteq_B \\
= & \quad \{\ \exists \text{ functor }\} \\
 & \exists h^\circ ; \exists k ; \sqsupseteq_B
\end{array}$$

$$
\begin{array}{ll}
= & \{\ \text{lemmas 6.5.5.1 and 6.5.5.2}\ \} \\
 & \exists\, h^{\circ} \,;\, \sqsupseteq_C \,;\, \exists\, k \\
= & \{\ \text{lemma 6.5.5.3}\ \} \\
 & \sqsupseteq_A \,;\, (\exists\, h)^{\circ} \,;\, \exists\, k \\
= & \{\ \text{def. } (\text{-})^{*}\ \} \\
 & \sqsupseteq_A \,;\, \exists^{*} R
\end{array}
$$

6.5.5.1 Lemma

$$\exists\, R \,;\, \sqsupseteq_B \ \supset\ \sqsupseteq_A \,;\, \exists\, R$$

Proof

$$
\begin{array}{ll}
 & \sqsupseteq_A \,;\, \exists\, R \ \subset\ \exists\, R \,;\, \sqsupseteq_B \\
\Leftrightarrow & \{\ \text{def. } \sqsupseteq\text{, division calculus (prop. 6.4.2)}\ \} \\
 & \sqsupseteq_A \,;\, \exists\, R \ \subset\ (\exists\, R \,;\, \ni_B)/\ni_B \\
\Leftrightarrow & \{\ \text{right-division}\ \} \\
 & \sqsupseteq_A \,;\, \exists\, R \,;\, \ni_B \ \subset\ \exists\, R \,;\, \ni_B \\
\Leftrightarrow & \{\ \ni : \exists \rightarrow \mathit{Rel}(\mathcal{E})\ \} \\
 & \sqsupseteq_A \,;\, \ni_A \,;\, R \ \subset\ \ni_A \,;\, R \\
\Leftrightarrow & \{\ \text{division calculus } (\sqsupseteq \,;\, \ni \,=\, \ni)\ \} \\
 & \text{true}
\end{array}
$$

6.5.5.2 Lemma

$$\exists\, h ; \sqsupseteq_B \ \subset\ \sqsupseteq_A \,;\, \exists\, h$$

Proof Let $\langle f, g\rangle$ be a tabulation of $\exists\, h \,;\, \sqsupseteq_B$. Define

$$z \;=\; \Lambda(f \,;\, \ni_A \cap g \,;\, \ni_B \,;\, h^{\circ})\ .$$

We have

$$\exists\, h \,;\, \sqsupseteq_B$$

$$\begin{array}{ll}
= & \{\ \langle f,g\rangle \text{ tabulation }\ \} \\
 & f^\circ\,;g \\
\subset & \{\ z \text{ entire }\ \} \\
 & f^\circ\,;z\,;z^\circ\,;g \\
\subset & \{\ \text{see below}\ \} \\
 & \sqsupseteq_A\,;\exists\,h
\end{array}$$

In the last step of this calculation, it was claimed that

$$f^\circ\,;z\ \subset\ \sqsupseteq_A \quad\text{and}\quad z^\circ\,;g\ \subset\ \exists\,h\ .$$

We deal with each of these proof obligations in tnrn. The first claim is equivalent to

$$f^\circ\,;z\,;\ni_A\ \subset\ \ni_A$$

by definition of $\sqsupseteq$. This last inclusion can be proved as follows:

$$\begin{array}{ll}
 & f^\circ\,;z\,;\ni_A \\
= & \{\ \text{def. } z\text{, power adjunction}\ \} \\
 & f^\circ\,;(f\,;\ni_A\ \cap\ g\,;\ni_B\,;h^\circ) \\
\subset & \{\ \text{monotonicity}\ \} \\
 & f^\circ\,;f\,;\ni_A \\
\subset & \{\ f \text{ simple}\ \} \\
 & \ni_A
\end{array}$$

To prove the second claim,

$$z^\circ\,;g\ \subset\ \exists\,h\ ,$$

one may reason

$$\begin{array}{ll}
 & z^\circ\,;g\ \subset\ \exists\,h \\
\Leftrightarrow & \{\ z \text{ map}\ \} \\
 & g\subset z\,;\exists\,h \\
\Leftrightarrow & \{\ \text{inclusion of maps}\ \} \\
 & g=z\,;\exists\,h
\end{array}$$

$\Leftrightarrow$ { def. z, power adjunction }

$g\,;\ni_B \;=\; (f\,;\ni_A \cap g\,;\ni_B\,;h^\circ)\,;h$

$\Leftrightarrow$ { h map, modular law }

$g\,;\ni_B \;=\; f\,;\ni_A\,;h \cap g\,;\ni_B$

$\Leftrightarrow$ { $\cap$ greatest lower bound }

$g\,;\ni_B \;\subset\; f\,;\ni_A\,;h$

$\Leftrightarrow$ { f map }

$f^\circ\,;g\,;\ni_B \;\subset\; \ni_A\,;h$

$\Leftrightarrow$ { right-division }

$f^\circ\,;g \;\subset\; (\ni_A\,;h)/\ni_B$

$\Leftrightarrow$ { $\ni : \exists \to Rel(\mathcal{E})$ }

$f^\circ\,;g \;\subset\; (\exists\,h\,;\ni_B)/\ni_B$

$\Leftrightarrow$ { division calculus (prop. 6.4.2), def. $\sqsupseteq$ }

$f^\circ\,;g \;\subset\; \exists\,h\,;\sqsupseteq_B$

$\Leftrightarrow$ { $\langle f,g\rangle$ tabulation of $\exists\,h\,;\sqsupseteq_B$ }

true

6.5.5.3 Lemma

$$\exists\,h^\circ;\sqsupseteq_A \;=\; \sqsupseteq_B\,;(\exists\,h)^\circ$$

Proof

$\exists\,h^\circ\,;\sqsupseteq_A$

$=$ { def. $\sqsupseteq$, division calculus (6.4.2) }

$(\exists\,h^\circ\,;\ni_A)/\ni_A$

$=$ { $\ni : \exists \to Rel(\mathcal{E})$ }

$(\ni_B\,;h^\circ)/\ni_A$

$=$ { division calculus }

$\ni_B/(\ni_A\,;h)$

$$\begin{array}{ll} = & \{\ \ni : \exists \rightarrow \mathit{Rel}(\mathcal{E})\ \} \\ & \ni_B/(\exists h\,;\ni_B) \\ = & \{\ \text{division calculus, def. } \sqsupseteq\ \} \\ & \sqsupseteq_B\,;(\exists h)^\circ \end{array}$$

6.5.6 Let $R : A \rightarrow B$ be a relation in $\mathit{Rel}(\mathcal{E})$. There are three obvious ways of extending R to power objects:

$$\mathrm{S}R \;=\; \in_A \backslash (R\,;\in_B) \tag{6.25}$$

$$\mathrm{H}R \;=\; (\ni_A\,;R)/\ni_B \tag{6.26}$$

$$\mathrm{M}R \;=\; \mathrm{S}R \cap \mathrm{H}R\,. \tag{6.27}$$

These extensions of R are named the *Smyth*, *Hoare*, and *Egli-Milner* extension of R, respectively. It follows from the properties of division that the Hoare extension could also be defined as follows:

$$\mathrm{H}R \;=\; \exists R\,;\sqsupseteq_B\,. \tag{6.28}$$

In fact, a similar characterisation could be given for the Smyth extension, for there is a simple relationship between the two:

$$\mathrm{S}R \;=\; (\mathrm{H}R^\circ)^\circ\,. \tag{6.29}$$

6.5.7 Proposition *For all $R : A \rightarrow B$,*

$$\exists^* R \;=\; \mathrm{M}R\,.$$

Proof To show that $\exists^* R \subset \mathrm{M}R$, one may reason as follows:

$$\begin{array}{ll} & \mathrm{M}R \\ = & \{\ \text{def. M}\ \} \\ & \mathrm{H}R \cap \mathrm{S}R \\ = & \{\ \text{equations (6.28) and (6.29)}\ \} \\ & (\exists R\,;\sqsupseteq_B) \cap (\exists R^\circ\,;\sqsupseteq_A)^\circ \\ = & \{\ \text{theorem 6.5.5}\ \} \\ & (\sqsupseteq_A\,;\exists^* R) \cap (\sqsupseteq_B\,;\exists^* R^\circ)^\circ \end{array}$$

$$
\begin{array}{ll}
\supset & \{ \ \sqsupseteq \text{ is reflexive } \} \\
& \exists^* R \cap (\exists^* R^\circ)^\circ \\
= & \{ \ (\text{-})^\circ \text{ involution } \} \\
& \exists^* R
\end{array}
$$

The proof of the reverse containment,

$$\exists^* R \subset \mathrm{M}R \ ,$$

is very similar to the proof of lemma 6.5.5.3, and will be omitted.

6.5.8 Proposition *A relation R is entire iff $\exists R \subset \exists^* R$.*

Proof By proposition 6.5.7, it suffices to show that

$$\exists R \subset \mathrm{H}R \quad \text{and} \quad \exists R \subset \mathrm{S}R$$

iff R is entire. Note that $\exists R \subset \mathrm{H}R$ is always true, since $\mathrm{H}R = \exists R \, ; \sqsupseteq_B$, and $\sqsupseteq_B$ is reflexive. Furthermore,

$$
\begin{array}{ll}
& \exists R \ \subset \ \mathrm{S}R \\
\Leftrightarrow & \{ \ \text{def. Smyth extension } \} \\
& \in_A ; \exists R \subset R \, ; \in_B \\
\Leftrightarrow & \{ \ (\exists R) \text{ map } \} \\
& \in_A \subset R \, ; \in_B ; (\exists R)^\circ \\
\Leftrightarrow & \{ \ \text{reciprocal, } \ni : \exists \rightarrow \mathit{Rel}(\mathcal{E}) \ \} \\
& \in_A \subset R \, ; R^\circ ; \in_A \\
\Leftrightarrow & \{ \ \in_A ; \{\text{-}\}_A^\circ = A \ \} \\
& A \subset R \, ; R^\circ
\end{array}
$$

7 Representing Partial Relations

It is a celebrated result of Lawvere and Tierney tbat a topos is coreflective in its category of simple relations. This chapter reports on an attempt to achieve a similar result for arbitrary relations. That is, it seeks to show that every relation in a topos can be made entire in some canonical way. Unfortunately, my attempt has been unsuccesful, and I only succeeded in establishing the desired theorem for Boolean toposes. Hopefully, the applications that are described in later chapters will provide an incentive for others to find a more satisfactory solution.

This chapter starts off by reviewing the theorem of Lawvere and Tierney about simple relations. It is then shown how their notion of a *simple–relation classifier* can be generalised to the concept of a *relation totaliser*. Every topos has a simple-relation classifier, but not every topos has a relation totaliser. We consider a particular counter-example, namely the topos of commuting squares, in some detail. It is then proved that every Boolean topos does have a relation totaliser, and we record some algebraic properties of relation totalisers for later use.

7.1 Simple–Relation Classifiers

7.1.1 In *Set*, there exists an obvious way of making a partial function entire, namely by adding an extra element to its target. Let us briefly review that construction before presenting its generalisation to an arbitrary topos. Consider a partial function $F : A \rightarrow B$. The aim is to turn F into a total functiou $\tilde{F} : A \rightarrow EB$. The set EB is the collection of all subsets of B that have at most one element:

$$EB \;=\; \{\, \{b\} \mid b \in B \,\} \cup \{\emptyset\}\ .$$

There exists an obvious injection $\eta_B : B \longleftarrow EB$ which maps every element of B to a singleton set. The function $\tilde{F} : A \rightarrow EB$ can be defined as follows:

$$\tilde{F}a \;=\; \begin{cases} \{F(a)\} & \text{if } a({<}F)a \\ \emptyset & \text{otherwise}\ . \end{cases}$$

It is characterised among other functions $A \rightarrow EB$ by the property that

$$\tilde{F}\,;\eta_B^{\circ} \;=\; F$$

in the category of relations $Rel(\mathcal{E})$.

7.1.2 Consider the category of relations over a regular category $\mathcal{E}$. A *simple-relation classifier* is a collection of monic maps

$$\eta_A : A \rightarrowtail EA$$

which satisfies the following universal property: for every simple relation $F : A \to B$ there exists a unique map

$$\widetilde{F} : A \to EB$$

such that

$$\widetilde{F} ; \eta_B^\circ \;=\; F \,.$$

For any regular category $\mathcal{E}$, the simple relations form a subcategory of $\mathit{Rel}(\mathcal{E})$, which we shall denote by $\mathit{Simple}(\mathcal{E})$. If $\mathcal{E}$ has simple-relation classifier η, the embedding of $\mathcal{E}$ into $\mathit{Simple}(\mathcal{E})$ has a right adjoint, and the unit of that adjunction is η. The right adjoint is given by

$$EF \;=\; (\eta_A^\circ ; F)^{\sim} \,.$$

7.1.3 Theorem (Lawvere–Tierney see e.g. [56], pp. 28–29) *A topos has a simple-relation classifier.*

Proof Although the proof of this theorem is well-known, I cannot resist showing you my own proof, which makes use of the calculus of relations. Consider the tabulation of $PB \cap \Lambda\{\text{-}\}_B^\circ$, say

$$\begin{tikzcd}
 & EB \arrow[dl, tail, "i_B"'] \arrow[dr, tail, "i_B"] & \\
PB & & PB
\end{tikzcd}$$

Let F be a simple relation. We wish to show the existence of a map $\widetilde{F}$ such that

$$\begin{tikzcd}
EB \arrow[r, tail, "i_B"] & PB \\
B \arrow[u, "\widetilde{F}"] \arrow[ur, "\Lambda F"'] &
\end{tikzcd}$$

commutes. Note that such a map $\widetilde{F}$ is necessarily unique, as i_B is monic. Furthermore, its existence is equivalent to the containment

$$\Lambda^\circ F ; \Lambda F \;\subset\; PB \cap \Lambda\{\text{-}\}_B^\circ$$

$\Leftrightarrow$ $\{$ ΛF map, $\cap$ greatest lower bound $\}$

$\Lambda F \subset \Lambda F ; \Lambda\{\text{-}\}_B^\circ$

$\Leftrightarrow$ $\{$ let $\langle d, f\rangle$ be a tabulation of F, power adjunction $\}$

$d^\circ ; f \subset \Lambda(d^\circ ; f) ; \{\text{-}\}_B^\circ$

$\Leftrightarrow$ $\{$ d, $\{\text{-}\}_B$ maps $\}$

$f ; \{\text{-}\}_B = d ; \Lambda(d^\circ ; f)$

$\Leftrightarrow$ $\{$ power adjunction, d monic $\}$

true

This proves the existence of $\widetilde{F}$. Define $\eta_B = \widetilde{B}$. Then η_B is monic because $\{\text{-}\}_B$ is monic. To show that η is a simple–relation classifier, it remains to prove that

$$\widetilde{F} ; i_B = \Lambda F \quad \text{iff} \quad \widetilde{F} ; \eta_B^\circ = F \ .$$

By the existence of the power adjunction, it suffices to show that

$$\eta_B \ = \ \in_B ; i_B^\circ \ .$$

We first prove the inclusion ($\subset$):

$\eta_B \subset \in_B ; i_B^\circ$

$\Leftrightarrow$ $\{$ reciprocal, η_B map $\}$

$\eta_B ; i_B ; \ni_B \supset B$

$\Leftarrow$ $\{$ power adjunction $\}$

$\eta_B ; i_B = \{\text{-}\}_B$

$\Leftrightarrow$ $\{$ def. η_B $\}$

true

For the other containment,

$$\eta_B \supset \in_B ; i_B^\circ$$

first observe that $\eta_B ; i_B = \{\text{-}\}_B$ implies $\eta_B = \{\text{-}\}_B ; i_B^\circ$ because i_B is monic. Using this fact, we reason:

$$\in_B ; i_B^\circ \subset \eta_B$$

$$\begin{aligned}
&\Leftrightarrow \quad \{\ \text{fact above}\ \} \\
&\quad\ \in_B ; i_B^\circ \ \subset\ \{\text{-}\}_B ; i_B^\circ \\
&\Leftrightarrow \quad \{\ i_B \text{ map}\ \} \\
&\quad\ \in_B ; i_B^\circ ; i_B \ \subset\ \{\text{-}\}_B \\
&\Leftarrow \quad \{\ \text{def. } i_B\ \} \\
&\quad\ \in_B ; (\Lambda\{\text{-}\}_B^\circ)^\circ \ \subset\ \{\text{-}\}_B \\
&\Leftrightarrow \quad \{\ \text{power adjunction}\ \} \\
&\quad\ \mathsf{true}
\end{aligned}$$

7.1.4 It follows that for any topos, we have au adjunction

$$(\,G, E, \eta, \eta^\circ\,)\ :\ \mathcal{E} \rightharpoonup \mathit{Simple}(\mathcal{E})\ .$$

This adjunction is in fact a restriction of the power adjunction. For consider a simple relation $F : A \rightarrow B$. By the construction in the preceding proof, the following diagram commutes:

$$\begin{tikzcd}
EA \arrow[r, "EF"] \arrow[d, leftrightarrow, "i_A"'] & EB \arrow[d, leftrightarrow, "i_B"] \\
PA \arrow[r, "\exists F"'] & PB
\end{tikzcd}$$

In words, this means that E is a *subfunctor* of $\exists$. It will be useful to bear in mind that $\exists$ and E are similar: many of the algebraic properties of $\exists$ are shared by E.

7.2 Relation Totalisers

In the previous section, it was shown how every simple relation in a topos can be made entire. Does that construction generalise to relations? It certainly does in *Set*. Given $R : A \rightarrow B$, one may construct an entire relation $\widehat{R} : A \rightarrow EB$ by defining

$$\begin{aligned}
\widehat{R} \ = \ & \{\,(a, \{b\}) \mid a(R)b\,\}\ \cup \\
& \{\,(a, \emptyset) \mid \neg(a({<}R)a)\,\}\ .
\end{aligned}$$

This construction is a true generalisation of the simple–relation classifier, for if R is simple $\widehat{R} = \widetilde{R}$. This suggests that we could define $\widehat{R}$ by the same universal property we used for simple relations. Unfortunately, however, there may be many relations S that satisfy

$$S\,;\eta_B^\circ \;=\; R\,, \tag{7.1}$$

for S could map all elements of A to $\emptyset$, even those which are in the domain of R. We need an additional condition, which says that if a is in the domain of R, the statement $a(S)b$ implies $a(R\,;\eta_B)b$. This condition is expressed by the following containment:

$$^{<}R\,;S \;\subset\; R\,;\eta_B\;. \tag{7.2}$$

7.2.1 Consider the category of relations over a regular category. A *relation totaliser* is a collection of monic maps

$$\eta_A : A \rightarrowtail EA$$

which satisfies the following universal property: for every relation $R : A \rightarrow B$ there exists precisely one entire relation

$$\tilde{R} : A \rightarrow EB$$

such that

$$\tilde{R}\,;\eta_B^\circ \;=\; R \quad \text{and} \quad {}^{<}R\,;\tilde{R} \;\subset\; R\,;\eta_B\;.$$

Note that any two relation totalisers are isomorphic. Furthermore, if a category has both a simple–relation classifier $(\eta, \hat{-})$ and a relation totaliser $(\sigma, \tilde{-})$, the simple–relation classifier is also a relation totaliser: given $R : A \rightarrow B$, the entire relation $\tilde{R} : A \rightarrow EB$ can be constructed as

$$\tilde{R} \;=\; \hat{R}\,;\widetilde{\sigma_B^\circ}\;.$$

We may conclude that if both exist, a simple–relatiou classifier and a relation totaliser are isomorphic.

7.2.2 Not every topos has a relation totaliser. As a counter-example, consider the category of commuting squares $Set^{\rightarrow}$, which is defined as follows.

- The objects are arrows of Set.

- The arrows are pairs of arrows in Set. That is, if

$$\begin{tikzcd}
\bullet \arrow[r, "f"] \arrow[d, "h"'] & \bullet \arrow[d, "k"] \\
\bullet \arrow[r, "g"'] & \bullet
\end{tikzcd}$$

is a commuting square in Set,

$$(f, g) : h \rightarrow k$$

is an arrow of $Set^{\rightarrow}$.

- Composition is defined componentwise:

$$(f, g) ; (m, l) = (f ; m, g ; l) \ .$$

- Identity arrows in $Set^{\rightarrow}$ are pairs of identity arrows in Set.

The category of commuting squares is a topos. A detailed discussion of this fact may be found in the book by Goldblatt [46], pp. 86 – 88. The simple-relation classifier of $Set^{\rightarrow}$ is given by the following diagram, which depicts its component at k:

$$\begin{tikzcd}
A \arrow[r, "\iota_1 ; \eta_{A+B}", "+"{marking, pos=0.03}] \arrow[d, "k"'] & E(A+B) \arrow[d, "{E[k, B]}"] \\
B \arrow[r, "\eta_B"', "+"{marking, pos=0.03}] & EB
\end{tikzcd}$$

where η is tbe simple-relation classifier of Set.

The category of relations over $Set^{\rightarrow}$ may be described as the category of semi-commutative squares:

- The objects are arrows of Set.
- Tbe arrows are pairs of arrows in $Rel(Set)$. That is, if the diagram

$$\begin{tikzcd}
\bullet \arrow[r, "R"] \arrow[d, "h"'] \arrow[dr, phantom, "\supset"] & \bullet \arrow[d, "k"] \\
\bullet \arrow[r, "S"'] & \bullet
\end{tikzcd}$$

semi-commutes, the pair

$$(R, S) : h \rightarrow k$$

is an arrow in $Rel(Set^{\rightarrow})$.

– Composition is defined componentwise.

It can be shown that ${}^{<}(R,S) = ({}^{<}R, {}^{<}S)$. In particular, a relation (R,S) in *Rel*($Set^{\rightarrow}$) is entire if and only if R and S are entire.

The simple–relation classifier of $Set^{\rightarrow}$ is not a relation totaliser. For consider the arrow between identities

$$(R,S) : \{1\} \rightarrow \{0,1\}$$

in *Rel*($Set^{\rightarrow}$), where

$$R = \emptyset \;\text{ and }\; S = \{(1,0),(1,1)\}\ .$$

The obvious way of making this relation entire is

$$\begin{tikzcd}
\{1\} \arrow[r, "T"] \arrow[d, "id"'] \arrow[dr, phantom, "\supset"] & E(\{0,1\} + \{0,1\}) \arrow[d, "{E[id,id]}"] \\
\{1\} \arrow[r, "S\,;\,\eta"'] & E\{0,1\}
\end{tikzcd}$$

where $T = S\,;\,\iota_2\,;\,\eta$. However, one could also take

$$T \;=\; \{(1,\eta(\iota_2(0)))\}\ .$$

In either case

$$(T, S\,;\,\eta)\,;\,(\iota_1\,;\,\eta,\eta)^{\circ} = (R,S) \;\text{ and }\; {}^{<}(R,S)\,;\,(T,S\,;\,\eta) = (R,S)\,;\,(\iota_1\,;\,\eta,\eta)\ .$$

It follows that the simple–relation classifier $(\iota_1\,;\,\eta,\eta)$ is not a relation totaliser.

The conclusion of this counter–example is that not every topos has a relation totaliser. So far we developed our results for arbitrary toposes, but now it will be necessary to impose further restrictions. In what follows, we shall focus on *Boolean* toposes. In view of the applications in later chapters, this restriction is not too severe. From a mathematical viewpoint, however, it is an unsatisfactory step, and a possible alternative is discussed in chapter 10. At the time of writing, it is unknown whether the existence of a relation totaliser implies that a topos is Boolean.

7.2.3 A topos $\mathcal{E}$ is said to be *Boolean* if for every object A, the family of subobjects A^{*} is a Boolean algebra. Equivalently, one could say that all homsets of *Rel*($\mathcal{E}$) are Boolean algebras. That is, every relation $R : A \rightarrow B$ has a *complement* $\overline{R} : A \rightarrow B$, which is characterised by

$$R \cap \overline{R} = \emptyset \;\text{ and }\; R \cup \overline{R} = [A \times B]$$

($[A \times B]$ is the maximum morphism $A \rightarrow B$). This additional assumption makes it possible to draw on some well-established results about classical relation algebras. For instance, we can use the *shunting* rule

$$T \subset R \cup \overline{S} \;\Leftrightarrow\; S \cap T \subset R\,. \tag{7.3}$$

Furthermore, using the shunting rule and the modular law, one can prove *Schröder's rule*, which consists of the following two equivalences:

$$R\,;S \subset T \;\Leftrightarrow\; R^{\circ}\,;\overline{T} \subset \overline{S} \;\Leftrightarrow\; \overline{T}\,;S^{\circ} \subset \overline{R}$$

(see *e.g.* Backhouse *et al.* [6]). Schröder's rule plays an important rôle in proving properties of classical relation algebras [10, 80].

7.2.4 Proposition *In a Boolean topos,*

$$\iota_1 : A \rightarrow (A + \tau)$$

is a relation totaliser, and for all $R : A \rightarrow B$

$$\tilde{R} \;=\; R\,;\iota_1 \,\cup\, \overline{(R\,;!_B)}\,;\iota_2\,.$$

Proof Let $R : A \rightarrow B$ be a relation. By the construction of coproducts in *Rel*($\mathcal{E}$) (par. 6.1.16), any morphism $A \rightarrow B + \tau$ is of the form

$$S\,;\iota_1 \,\cup\, T\,;\iota_2$$

for some $S : A \rightarrow B$ and $T : A \rightarrow \tau$. Because the coproduct injections are disjoint monics, the identity

$$(S\,;\iota_1 \,\cup\, T\,;\iota_2)\,;\iota_1{}^{\circ} \;=\; R$$

is equivalent to $S = R$. Lemma 7.2.4.1 says that the inclusion

$$^{<}R\,;(R;\iota_1 \,\cup\, T;\iota_2) \;\subset\; R\,;\iota_1$$

is equivalent to $T \subset \overline{R\,;!_B}$. According to lemma 7.2.4.2, the reverse containment $\overline{R\,;!_B} \subset T$ holds if and only if

$$^{<}R \cup {}^{<}T \;=\; A\,.$$

Hence, if $(R;\iota_1 \cup T;\iota_2)$ is entire, $T = \overline{R\,;!_B}$. Lemma 7.2.4.3 shows that $(R;\iota_1 \cup T;\iota_2)$ is entire for $T = \overline{R\,;!_B}$.

7.2.4.1 Lemma

$$T \subset \overline{R ; !_B} \quad \Leftrightarrow \quad {}^{<}R ; (R ; \iota_1 \cup T ; \iota_2) \subset R ; \iota_1$$

Proof

$$T \subset \overline{R ; !_B}$$

$\Leftrightarrow$ { complement }

$$R ; !_B \subset \overline{T}$$

$\Leftrightarrow$ { prop. 5.4.8: ${}^{<}(R ; !_B) = {}^{<}R ; !_A$ and $!_A = \overline{\emptyset}$ }

$${}^{<}R ; \overline{\emptyset} \subset \overline{T}$$

$\Leftrightarrow$ { ${}^{<}R$ symmetric, Schröder's rule }

$${}^{<}R ; T \subset \emptyset$$

$\Leftrightarrow$ { injections are disjoint }

$${}^{<}R ; T ; \iota_2 \subset R ; \iota_1$$

$\Leftrightarrow$ { $\cup$ least upper bound }

$$R ; \iota_1 \cup {}^{<}R ; T ; \iota_2 \ \subset \ R ; \iota_1$$

$\Leftrightarrow$ { ; distributes over $\cup$ }

$${}^{<}R ; (R ; \iota_1 \cup T ; \iota_2) \ \subset \ R ; \iota_1$$

7.2.4.2 Lemma

$${}^{<}R \cup {}^{<}T = A \quad \Leftrightarrow \quad \overline{R ; !_B} \subset T$$

Proof

$$\overline{R ; !_B} \subset T$$

$\Leftrightarrow$ { prop. 5.4.8 }

$$\overline{R ; !_B} \cap !_A \subset T$$

$\Leftrightarrow$ { shunting }

$$R ; !_B \cup T \supset !_A$$

$$\begin{aligned}
\Leftrightarrow &\quad \{\ \text{prop. 5.4.8}\ \} \\
& {<}R\,;!_A\ \cup\ {<}T\,;!_A = !_A \\
\Leftrightarrow &\quad \{\ ;\ \text{distributes over}\ \cup\ \} \\
& ({<}R \cup {<}T)\,;!_A = !_A \\
\Leftrightarrow &\quad \{\ \text{prop. 5.4.8}\ \} \\
& {<}R \cup {<}T = A
\end{aligned}$$

7.2.4.3 Lemma

$$ {<}R \cup {<}(\overline{R\,;!_B}) \;=\; A $$

Proof First note that $\overline{R\,;!_B} = \overline{R};!_B$: the containment ($\supset$) follows from Schröder's rule and the fact that $!_B$ is a map, and the reverse inclusion ($\subset$) is a consequence of the shunting rule. The lemma can now be proved by a simple calculation:

$$\begin{aligned}
& {<}R \cup {<}(\overline{R\,;!_B}) \\
= &\quad \{\ \text{above, prop. 5.4.8}\ \} \\
& {<}R \cup {<}\overline{R} \\
= &\quad \{\ \text{domain distributes over union}\ \} \\
& {<}(R \cup \overline{R}) \\
= &\quad \{\ \text{complement}\ \} \\
& {<}[A \times B] \\
= &\quad \{\ \text{maximum morphism is entire}\ \} \\
& A
\end{aligned}$$

7.2.5 As an immediate consequence of the above construction, we obtain the following monotonicity property of the $(\text{-})^{\sim}$ operator:

Proposition *Let $\mathcal{E}$ be a Boolean topos. Suppose that ${<}R = {<}S$. Then*

$$ R \subset S \;\Leftrightarrow\; \bar{R} \subset \tilde{S}\,. $$

One could also prove this lemma for any topos with a relation totaliser, but the argument is unduly laborious.

7.3 Properties of Relation Totalisers

The construction of a relation totaliser in a Boolean topos is not suitable for use in calculational proofs. In a sense, it introduces a case distinction between whether an element is in the domain of a relation or not. Van Gasteren [95] argues convincingly that such case analyses should he avoided if one is aiming for purely syntactic proofs. Therefore, in this section, we shall develop a calculus which only depends on the definition of a relation totaliser. Throughout this section, it is assumed that we are working in a topos that has a relation totaliser.

7.3.1 The next proposition says that the requirement $^{<}R \,;\, \widetilde{R} \subset R \,;\, \eta_B$ in the definition of *relation totaliser* can be replaced by $^{<}(\widetilde{R}^{\circ} \,;\, R) \subset \eta_B{}^{>}$. The latter is easier to check in practice. In the sequel, we shall use the more practical definition of *relation totaliser* without explicitly referring to this paragraph.

Proposition *Suppose that* $S \,;\, h^{\circ} \subset R$. *Then*

$$^{<}R \,;\, S \subset R \,;\, h \quad \Leftrightarrow \quad {}^{<}(S^{\circ} \,;\, R) \subset h^{>} \ .$$

Proof We prove the equivalence by mutual implication. The forward implication is a trivial application of the domain calculus (prop. 5.4.6). To prove the backward implication, one may reason as follows:

$$
\begin{array}{ll}
 & ^{<}(S^{\circ} \,;\, R) \subset h^{>} \\
\Leftrightarrow & \{ \text{ domain calculus (prop. 5.4.6) } \} \\
 & (^{<}R \,;\, S)^{>} \subset h^{>} \\
\Rightarrow & \{ \text{ pre-compose with } {}^{<}R \,;\, S \ \} \\
 & ^{<}R \,;\, S \subset {}^{<}R \,;\, S \,;\, h^{>} \\
\Leftrightarrow & \{ \text{ range of map } \} \\
 & ^{<}R \,;\, S \subset {}^{<}R \,;\, S \,;\, h^{\circ} \,;\, h \\
\Rightarrow & \{ \text{ assumption: } S \,;\, h^{\circ} \subset R \ \} \\
 & ^{<}R \,;\, S \subset R \,;\, h
\end{array}
$$

7.3.2 Proposition *Let $A \xrightarrow{R} B \xrightarrow{S} C$. Suppose that R is entire, and*

$$^{<}(R^\circ ; R ; S) \;\subset\; ^{<}S \;.$$

Then $(R ; S)^{\sim} = R ; \tilde{S}$.

Proof It is clear that $R ; \tilde{S}$ is entire, and that

$$R ; \tilde{S} ; \eta_C^\circ \;=\; R ; S \;.$$

It remains to show

$$^{<}((R ; \tilde{S})^\circ ; R ; S) \;\subset\; \eta_C{}^{>} \;.$$

Here is a proof:

$$\begin{aligned}
& ^{<}((R ; \tilde{S})^\circ ; R ; S) \\
= \quad & \{ \text{ reciprocal } \} \\
& ^{<}(\tilde{S}^\circ ; R^\circ ; R ; S) \\
\subset \quad & \{ \text{ assumption } ^{<}(R^\circ ; R ; S) \subset {}^{<}S \; \} \\
& ^{<}(\tilde{S}^\circ ; S) \\
\subset \quad & \{ \text{ relation totaliser } \} \\
& \eta_C{}^{>}
\end{aligned}$$

7.3.3 Recall the similarity between the power adjunction and the adjunction between a topos and its category of simple relations

$$(\, G, E, \eta, \eta^\circ \,) \;:\; \mathcal{E} \rightarrow \mathit{Simple}(\mathcal{E}) \;.$$

This adjunction defines a monad

$$(E, \eta, E\eta^\circ)$$

in $\mathcal{E}$. The proof of the next proposition makes use of the naturality of η.

Proposition *Let $A \xrightarrow{R} B \xrightarrow{h} C$. Then*

$$(R ; h)^{\sim} \;=\; \tilde{R} ; Eh \;.$$

Proof That $\tilde{R} ; Eh$ is entire is obvious; furthermore

$$\tilde{R} ; Eh ; \eta_C^{\circ} \;=\; \tilde{R} ; \eta_B^{\circ} ; h \;=\; R ; h \; .$$

The proof can now be completed as follows:

$$
\begin{array}{ll}
 & ^{<}((\tilde{R} ; Eh)^{\circ} ; R ; h) \\
= & \quad \{ \; h \text{ entire, reciprocal } \} \\
 & ^{<}((Eh)^{\circ} ; \tilde{R}^{\circ} ; R) \\
\subset & \quad \{ \text{ relation totaliser, domain calculus (prop. 5.4.6) } \} \\
 & ^{<}((Eh)^{\circ} ; \eta_B^{\circ}) \\
\subset & \quad \{ \text{ range is domain of reciprocal } \} \\
 & (\eta_B ; Eh)^{>} \\
= & \quad \{ \; \eta \text{ natural } \} \\
 & (h ; \eta_C)^{>} \\
\subset & \quad \{ \text{ domain calculus } \} \\
 & \eta_C{}^{>}
\end{array}
$$

7.3.4 Let $F : \mathcal{D} \rightarrow \mathcal{E}$ be a relator. In analogy with the cross–operator on F, one may define a natural transformation $\overline{F} : (F \circ E) \rightarrow (E \circ F)$:

$$\overline{F}_A \;=\; ((F\eta_A)^{\circ})^{\sim} \; .$$

This natural transformation is called the *smash on* F. Intuitively, it takes a structure (say a tuple), and it returns that tnple if all its components are non–fictitious:

$$F\eta_A ; \overline{F}_A \;=\; \eta_{FA} \; . \tag{7.4}$$

However, if one of the components is fictitious, it returns the fictitious valne. The term *smash* is inspired by the connection with *smash products* in programming language semantics.

Proposition *Let* $F : \mathcal{D} \rightarrow \mathcal{E}$ *be a relator. Let* $R : A \rightarrow B$ *be a relation in* $\mathcal{D}$. *Then*

$$(F^{\star} R)^{\sim} \;=\; F^{\star}\tilde{R} ; \overline{F}_B .$$

Proof That $(F^\star\widetilde{R} ; \overline{F}_B)$ is entire is obvious. Also,

$$F^\star\widetilde{R} ; \overline{F}_B ; \eta_B^\circ \;=\; F^\star\widetilde{R} ; (F\eta_B)^\circ \;=\; F^\star(\widetilde{R} ; \eta_B{}^\circ) \;=\; F^\star R\ .$$

It remains to calculate:

$$\begin{array}{ll}
 & {<}((F^\star\widetilde{R} ; \overline{F}_B)^\circ \;;\; F^\star R) \\
= & \quad \{\ \text{reciprocal, } F^\star \text{ functor}\ \} \\
 & {<}(\overline{F}^\circ_B ; F^\star(\widetilde{R}^\circ ; R)) \\
= & \quad \{\ \text{domain calculus (prop. 5.4.6)}\ \} \\
 & {<}(\overline{F}^\circ_B ; {<}(F^\star(\widetilde{R}^\circ ; R))) \\
= & \quad \{\ \text{relators preserve domain (prop. 5.4.7)}\ \} \\
 & {<}(\overline{F}^\circ_B ; F^\star{<}(\widetilde{R}^\circ ; R)) \\
\subset & \quad \{\ \text{relation totaliser}\ \} \\
 & {<}(\overline{F}^\circ_B ; F^\star\eta_B{>}) \\
= & \quad \{\ \text{range of map, def. } F^\star\ \} \\
 & {<}(\overline{F}^\circ_B ; (F\eta_B)^\circ ; F\eta_B) \\
= & \quad \{\ F\eta_B \text{ entire, domain calculus}\ \} \\
 & {<}(\overline{F}^\circ_B ; (F\eta_B)^\circ) \\
= & \quad \{\ \text{range is domain of reciprocal}\ \} \\
 & (F\eta_B ; \overline{F}_B){>} \\
= & \quad \{\ \text{equation (7.4)}\ \} \\
 & \eta_{FB}{>}
\end{array}$$

8 Maximisation in Preorders

Consider a set of finite sequences. Such a set may have multiple elements of minimum length, for the length ordering on sequences is not anti-symmetric. We shall write $x(\mathit{minlen})a$ if a is an element of x of minimum length. The relation *minlen* distributes over union in the following sense:

$$\bigcup ; \mathit{minlen} \;=\; \exists\, \mathit{minlen} \,;\, \mathit{minlen} \;.$$

Note that $\exists\,\mathit{minlen}$ is a set-valued function which returns all minimum elements of all components of its argument. Consequently, a computer program that evaluates the right-hand side of the above equation might be very time–consuming. This contrasts with the situation where $\exists$ is replaced by $\exists^{\star}$. The relation $\exists^{\star}\mathit{minlen}$ yields a *stratified sample* of the minimum elements, and here an implementation just needs to return some minimum value for each component of the argument set. But does the above equation still hold when $\exists$ is replaced by $\exists^{\star}$? In this chapter, we shall try to answer this question.

We start off by defining the notion of *maximum elements* in an arbitrary topos. This definition is a generalisation of the relation *minlen* discussed above; we consider maximum elements instead of minimum elements for technical reasons. It turns out that *minlen* does indeed distribute over union. Unfortunately, distributivity does not hold when $\exists$ is replaced by $\exists^{\star}$. This problem will be remedied by making use of relation totalisers. It is shown how one can make the *maximum element* relation entire in such a way that the desired distributivity property holds.

A different kind of distributivity arises in the following context. Write (:) for the binary operator that places an element at the front of a list:

$$a_0 : [a_1, a_2, \ldots, a_n] \;=\; [a_0, a_1, a_2, \ldots, a_n] \;.$$

This operator is commonly known as *cons*. Furthermore, write $(a{:})$ for the operator that places the element a at the front of a list. *Cons* distributes over *minlen* in the following sense:

$$\mathit{minlen} \;;\, (a{:}) \;=\; \exists(a{:})\,;\, \mathit{minlen} \;.$$

Or perhaps it does not? What about the case when the set of sequences is empty? These issues are addressed in the last section of this chapter. It is shown that certain monotonicity conditions imply distributivity.

8.1 Maximum Elements

8.1.1 Throughout this section, it is assumed that we are working in a topos $\mathcal{E}$. Let $R : A \rightarrow B$ be a relation. Define the *upper bound* relation of R by

$$\begin{array}{rcl} R\uparrow & : & PA \rightarrow B \\ R\uparrow & = & \in_A \backslash R \; . \end{array}$$

Translated into ordinary set theoretic notation, this definition reads as follows:

$$x(R\uparrow)b \;\; = \;\; \forall a \in A \; : \; a {\in_A} x \; \Rightarrow \; a(R)b \; . \qquad (8.1)$$

Note that left-division may be characterised in terms of the upper bound operator:

$$S\backslash R \;\; = \;\; \Lambda S^\circ \, ; R\uparrow \; . \qquad (8.2)$$

The proof of this property is a simple application of the division calculus. It has two consequences that will be useful in later calculations. The first shows how the upper bonnd relation distributes over union:

$$\bigcup \, ; R\uparrow \;\; = \;\; (R\uparrow)\uparrow \; . \qquad (8.3)$$

The second corollary tells how an existential image followed by an upper bound yields another npper bound:

$$\exists S \, ; R\uparrow \;\; = \;\; (S^\circ \backslash R)\uparrow \; . \qquad (8.4)$$

8.1.2 Let $R : A \rightarrow A$ in $\mathit{Rel}(\mathcal{E})$. Define the *maximum* relation of R by

$$\begin{array}{rcl} \mathit{max} R & : & PA \rightarrow A \\ \mathit{max} R & = & R\uparrow \cap \ni_A \; . \end{array} \qquad (8.5)$$

This definition is in accordance with the usual set theoretic notion of maximum elements: a is a maximum element of x if it is an upper bound of x and it is an element of x. The properties of the upper bound relation translate into facts about the maximum relation. For example, it is immediate from equation (8.2) that

$$\Lambda S \, ; \mathit{max} R \;\; = \;\; S^\circ \backslash R \cap S \; . \qquad (8.6)$$

In particnlar, we have the following instance of equation (8.4):

$$\exists S \, ; \mathit{max} R \;\; = \;\; (S^\circ \backslash R)\uparrow \, \cap \, \ni_A ; S \; . \qquad (8.7)$$

8.1.3 Before we can continue our exploration of the properties of $\mathit{max}\,R$, it will be necessary to settle some terminology. A relation $R : A \rightarrow A$ is said to be *transitive* if $R \, ; R \subset R$. It is called *reflexive* if $A \subset R$. A relatiou that is both reflexive and transitive is said to be a *preorder*. Preorders frequently arise in optimisation problems. A typical example is the length preorder on finite sequences, which is defined as follows:

$$\leq_{\mathit{len}} \;=\; \mathit{length} \; ; (\leq) \, ; \mathit{length}^{\circ} \; .$$

Here *length* is the function that returns the length of a sequence, and $(\leq)$ is the standard order on natural numbers. The relation *minlen* that we discussed in the introduction to this chapter is given by $\mathit{max}(\geq_{\mathit{len}})$.

A relation $R : A \rightarrow A$ is called *anti-symmetric* if $R^{\circ} \cap R = A$. An anti-symmetric preorder is said to be a *partial order*. The standard order on natnral numbers is anti-symmetric, but the length preorder on lists is not.

Finally, a relation $R : A \rightarrow A$ is said to be *well-bounded* if

$${}^{<}\ni_A \subset {}^{<}(\mathit{max}\,R) \; .$$

(The reverse inclusion ${}^{<}(\mathit{max}\,R) \subset {}^{<}\ni_A$ is always satisfied.) While preorders and partial orders can he defined in any regular category, the definition of well-boundedness makes essential use of the topos structure. The reciprocal of the length preorder on sequences is well-bounded: every non-empty set of seqnences contains a sequence of minimum length.

8.1.4 Proposition *Let $R : A \rightarrow A$ be a reflexive relation. Then*

$$\mathit{max}^{\circ} R \, ; \mathit{max}\,R \;=\; R \cap R^{\circ} \; .$$

Therefore, $(\mathit{max}\,R)$ is simple iff R is anti-symmetric.

Proof We aim to prove the above identity by mutual inclusion. The containment $(\subset)$ follows by unfolding the definitions:

$$\mathit{max}^{\circ} R \, ; \mathit{max}\,R \;\subset\; \in_A ; R\!\upharpoonright \;\subset\; R \; .$$

Taking the reciprocal on both sides of this inequation, one also obtains that $\mathit{max}^{\circ} R$; $\mathit{max}\,R \subset R^{\circ}$. Therefore, one may conclude that

$$\mathit{max}^{\circ} R \, ; \mathit{max}\,R \;\subset\; R \cap R^{\circ} \; .$$

To prove the reverse inclusion, let (h, k) be a tabulation of $R \cap R^{\circ}$, and define $z = \Lambda(h \cup k)$. One may reason as follows:

$$R \cap R^{\circ} \;=\; h^{\circ} \, ; k \;\subset\; h^{\circ} \, ; z \, ; z^{\circ} \, ; k \;\subset\; \mathit{max}^{\circ} R \, ; \mathit{max}\,R \; .$$

Only the last step of this calculation needs a proof. Since h and k are symmetrical, it suffices to show that

$$z^\circ ; k \subset max\, R \ .$$

Here is a proof:

$$
\begin{array}{ll}
 & z^\circ ; k \subset max\, R \\
\Leftrightarrow & \quad \{ \ z \text{ map } \} \\
 & k \subset z ; max\, R \\
\Leftrightarrow & \quad \{ \ \text{def. } z, \text{ maximum (eq. 8.6) } \} \\
 & k \ \subset \ (h^\circ \cup k^\circ)\backslash R \ \cap \ (h \cup k) \\
\Leftrightarrow & \quad \{ \ \cap \text{ greatest lower bound }, \cup \text{ upper bound } \} \\
 & k \subset (h^\circ \cup k^\circ)\backslash R \\
\Leftrightarrow & \quad \{ \ \text{left–division } \} \\
 & (h^\circ \cup k^\circ) ; k \ \subset \ R \\
\Leftrightarrow & \quad \{ \ \text{modular law } \} \\
 & h^\circ ; k \ \cup \ A \ \subset \ R \\
\Leftrightarrow & \quad \{ \ \cup \text{ least upper bound } \} \\
 & h^\circ ; k \ \subset \ R \ \text{ and } \ A \subset R \\
\Leftrightarrow & \quad \{ \ (h, k) \text{ tabulation of } R \cap R^\circ, \ R \text{ reflexive } \} \\
 & \text{true}
\end{array}
$$

8.1.5 Suppose we want to determine the length of a shortest element of a set of sequences. There are two ways of performing this task. In the first approach, one takes the problem statement literally: first find a shortest sequence, and then compute its length. In the second approach, one computes the length of all sequences in the set, and then takes its minimum. The next proposition shows that both methods produce the same result:

Proposition *Let* $B \xrightarrow{k} A \xrightarrow{R} A$. *Then*

$$max(k ; R ; k^\circ) ; k \ = \ \exists\, k ; max\, R \ .$$

Proof

$$\begin{array}{cl}
 & \exists\, k\,;\, \mathit{max}\, R \\
= & \{\ \text{maximum (eq. 8.7)}\ \} \\
 & (k^{\circ}\backslash R){\uparrow}\ \cap\ \ni_B\,;\, k \\
= & \{\ \text{division calculus (prop. 6.4.2)}\ \} \\
 & (k\,;\, R){\uparrow}\ \cap\ \ni_B\,;\, k \\
= & \{\ \text{modular law}\ \} \\
 & ((k\,;\, R){\uparrow}\,;\, k^{\circ}\ \cap\ \ni_B)\,;\, k \\
= & \{\ \text{def. upper bound (eq. 8.1), division calculus}\ \} \\
 & ((k\,;\, R\,;\, k^{\circ}){\uparrow}\ \cap\ \ni_B)\,;\, k \\
= & \{\ \text{def. maximum}\ \} \\
 & \mathit{max}(k\,;\, R\,;\, k^{\circ})\,;\, k
\end{array}$$

8.1.6 When we defined the concept of well–houndedness, we noted that the reciprocal of the length preorder on lists is well–bouuded. Not surprisingly, this fact generalises to arbitrary preorders of the form $(k\,;R\,;k^{\circ})$, where R is well–bounded.

Proposition *Let* $B \xrightarrow{k} A \xrightarrow{R} A$. *If* R *is a well–bounded preorder,*

$$k\,;R\,;k^{\circ}$$

is a well–bounded preorder as well.

Proof That $(k\,;R\,;k^{\circ})$ is a preorder is evident. That it is also well–hounded may be proved as follows:

$$\begin{array}{cl}
 & {}^{<}(\mathit{max}(k\,;\, R\,;\, k^{\circ})) \\
= & \{\ k\ \text{entire, domain calculus (prop. 5.4.6)}\ \} \\
 & {}^{<}(\mathit{max}(k\,;\, R\,;\, k^{\circ})\,;\, k) \\
= & \{\ \text{prop. 8.1.5}\ \} \\
 & {}^{<}(\exists\, k\,;\, \mathit{max}\, R) \\
= & \{\ R\ \text{well–bounded, domain calculus}\ \} \\
 & {}^{<}(\exists\, k\,;\, \ni_A)
\end{array}$$

$=$ $\{\ \ni : \exists \rightarrow \mathit{Rel}(\mathcal{E})\ \}$

$^{<}(\ni_B ; \boldsymbol{k})$

$=$ $\{\ \boldsymbol{k}$ entire, domain calculus $\}$

$^{<}\ni_B$

8.1.7 Consider a collection of sets of sequences. To find a shortest sequence of all the components, one can first take the union, and then find a shortest sequence. Alternatively, one might take all shortest elements of each component set, and then select a shortest from those candidates. The next theorem is a precise statement of this fact. Note that the theorem is only applicable to relations that are well–bounded and transitive.

Theorem *Let $R : A \rightarrow A$ be well–bounded and transitive. Then*

$$\bigcup_A ; \mathit{max}R \ = \ \exists\, \mathit{max}R ; \mathit{max}R \ .$$

Proof

$$\bigcup_A ; \mathit{max}R$$

$=$ $\{$ maximum (eq. 8.7), $\bigcup_A = \exists \ni_A$ $\}$

$$(\ni_A \backslash R)\uparrow \cap (\ni_{PA} ; \ni_A)$$

$=$ $\{$ def. upper bound (eq. 8.1) $\}$

$$(R\uparrow)\uparrow \cap (\ni_{PA} ; \ni_A)$$

$=$ $\{$ def. upper bound $\}$

$$(\in_{PA} \backslash (R\uparrow)) \cap (\ni_{PA} ; \ni_A)$$

$=$ $\{$ division calculus (prop. 6.4.2) $\}$

$$(\in_{PA} \backslash (R\uparrow)) \cap (\ni_{PA} ; (R\uparrow \cap \ni_A))$$

$=$ $\{$ defs. upper bound and maximum $\}$

$$(R\uparrow)\uparrow \cap (\ni_{PA} ; \mathit{max}R)$$

$=$ $\{$ lemma 8.1.7.1 $\}$

$$(\exists\, \mathit{max}R ; R\uparrow) \cap (\ni_{PA} ; \mathit{max}R)$$

$$\begin{aligned}
= &\quad \{\ \ni\ :\ \exists \rightarrow Rel(\mathcal{E})\ \} \\
&(\exists\, max R\, ;\, R{\uparrow}) \cap (\exists\, max R\, ;\, \ni_A) \\
= &\quad \{\ \exists\, max R \text{ simple }\} \\
&\exists\, max R\, ;\, (R{\uparrow} \cap \ni_A) \\
= &\quad \{\ \text{def. maximum}\ \} \\
&\exists\, max R\, ;\, max R
\end{aligned}$$

8.1.7.1 Lemma *Let $R : A \rightarrow A$ bc transitive and well-bounded. Then*

$$\exists\, max R\, ;\, R{\uparrow} \;=\; (R{\uparrow}){\uparrow}\ .$$

Proof

$$\begin{aligned}
&\exists\, max R\, ;\, R{\uparrow} \\
= &\quad \{\ \text{upper bound (eq. 8.4)}\ \} \\
&((max^{\circ} R)\backslash R){\uparrow} \\
= &\quad \{\ \text{below}\ \} \\
&(R{\uparrow}){\uparrow}
\end{aligned}$$

In the last step, it was claimed that

$$R{\uparrow} \;=\; (max^{\circ} R)\backslash R\ .$$

We aim to prove this identity by mutual containment. First observe that

$$\begin{aligned}
&R{\uparrow} \subset (max^{\circ} R)\backslash R \\
\Leftrightarrow &\quad \{\ \text{def. upper bound}\ \} \\
&\in_A\backslash R \subset (max^{\circ} R)\backslash R \\
\Leftarrow &\quad \{\ (\text{-})\backslash R \text{ anti-monotonic}\ \} \\
&(max^{\circ} R) \subset \in_A \\
\Leftrightarrow &\quad \{\ \text{def. maximum}\ \} \\
&\text{true}
\end{aligned}$$

To prove the other containment, note that it is equivalent to

$$\in_A ; (max^\circ R)\backslash R \ \subset \ R \ ,$$

by definition of upper bound and left-division. The proof can be completed as follows:

$$\begin{array}{ll}
 & \in_A ; (max^\circ R)\backslash R \\
= & \quad \{ \ R \text{ well-bounded } \} \\
 & \in_A ; {}^{<}(max R) ; (max^\circ R)\backslash R \\
= & \quad \{ \ \text{def. strict left-division (par. 6.4.6)} \ \} \\
 & \in_A ; (max^\circ R) \backslash\backslash R \\
\subset & \quad \{ \ \text{strict left-division (eq. 6.21)} \ \} \\
 & \in_A ; max R ; R \\
\subset & \quad \{ \ \text{def. maximum} \ \} \\
 & \in_A ; R{\uparrow} ; R \\
\subset & \quad \{ \ \text{def. upper bound} \ \} \\
 & R ; R \\
\subset & \quad \{ \ R \text{ transitive } \} \\
 & R
\end{array}$$

8.2 Selectors

The theorem which says that $\bigcup ; max R = \exists\, max R ; max R$ is unsatisfactory in the following sense. What we would really like to prove is this: to select some optimal element from the union of a collection of sets, it suffices to select *some* optimal element from each component set, and then select some optimal element from that set of candidates. In contrast, tbe above theorem says that we should find *all* optimal elements of each component set.

8.2.1 A relation $T : PA \to A$ is said to be a *selector* if

$$\bigcup_A ; T = \exists^* T ; T \quad \text{and} \quad \{\text{-}\}_A ; T = A \ .$$

The first requirement states the property that we discussed in the preceding paragraph. The second requirement says that if you select an element from a singleton set, there is only one choice, namely the single element.

The *maximum* relation is not a selector, because the empty set does not have a maximum element. Consequently, the inclusion

$$\bigcup_A \,;\, maxR \;\subset\; \exists^{*} maxR \,;\, maxR$$

does not hold. This problem might be solved on an ad hoc basis, by introducing a fictitious value that is smaller than all other values ($-\infty$). The maximum element of the empty set is then defined to be $-\infty$. Using relation totalisers, the trick of introducing $-\infty$ can be presented in systematic way. Throughout this section, it is assumed that we are working in a Boolean topos.

8.2.2 Let $R : A \rightarrow A$ be a relation. Define the *selector* of R by

$$\begin{array}{lcl} selR & : & PEA \rightarrow EA \\ selR & = & \exists\eta_A^{\circ} \,;\, \widetilde{max}R \,. \end{array} \tag{8.8}$$

This definition is inspired by the informal considerations sketched above: $selR$ maps its argument set to a maximum element. If such a maximum element does not exist, it returns a fictitious value, which plays the rôle of minus iufinity ($-\infty$).

8.2.3 The next theorem states the anticipated result, namely that $selR$ is a selector. The proof of the theorem is unduly complicated and lengthy. However, as it seems awkward to verify this theorem in ordinary set theory, I was glad to find a proof at all. Hopefully, if the theorem turns out to be useful in practice, others will find more elegant ways of presenting its proof.

Theorem *Suppose that R is a well-bounded preorder. Then* $(sel\,R)$ *is a selector.*

Proof We shall present the proof in a sequence of five lemmas. Lemma 8.2.3.1 shows that

$$\{\cdot\}_{EA} \,;\, selR \;=\; EA \;.$$

Lemma 8.2.3.2 says that

$$\bigcup_{EA} \,;\, selR \;=\; \exists\, selR \,;\, selR \;.$$

Together with the fact that $selR$ is entire and proposition 6.5.8, this yields the containment

$$\bigcup_{EA} \,;\, selR \;\subset\; \exists^{*} selR \,;\, selR \;.$$

The reverse containment follows from lemmas 8.2.3.2 and 8.2.3.5.

8.2.3.1 Lemma *If R is reflexive, then*

$$\{\cdot\}_{EA} \,;\, selR \;=\; EA\;.$$

Proof

$$
\begin{array}{ll}
 & \{\cdot\}_{EA} \,;\, \exists\,\eta_A^\circ \,;\, \widetilde{max}\,R \\
= & \quad \{ \text{ power adjunction } \} \\
 & \Lambda\eta_A^\circ \,;\, \widetilde{max}\,R \\
= & \quad \{ \text{ } \Lambda\eta_A^\circ \text{ map, relation totaliser (prop. 7.3.2) } \} \\
 & (\Lambda\eta_A^\circ \,;\, max\,R)^\sim \\
= & \quad \{ \text{ maximum (eq. 8.6) } \} \\
 & ((\eta_A\backslash R) \cap \eta_A^\circ)^\sim \\
= & \quad \{ \text{ see below } \} \\
 & \widetilde{\eta_A^\circ} \\
= & \quad \{ \text{ relation totaliser } \} \\
 & EA
\end{array}
$$

Here is a proof of the penultimate step:

$$
\begin{array}{ll}
 & \eta_A\backslash R \cap \eta_A^\circ \;=\; \eta_A^\circ \\
\Leftrightarrow & \quad \{ \text{ } \cap \text{ greatest lower bound } \} \\
 & \eta_A^\circ \subset \eta_A\backslash R \\
\Leftrightarrow & \quad \{ \text{ left division } \} \\
 & \eta_A \,;\, \eta_A^\circ \subset R \\
\Leftrightarrow & \quad \{ \text{ } \eta_A \text{ monic } \} \\
 & A \subset R \\
\Leftrightarrow & \quad \{ \text{ } R \text{ reflexive } \} \\
 & \text{true}
\end{array}
$$

8.2.3.2 The algebraic properties of *selR* are very similar to those of *maxR*. In paragraph 8.1.7, it was shown that

$$\bigcup_A \,;\, max\,R \;=\; \exists\, max\,R \,;\, max\,R \;,$$

provided R is well-bounded and transitive. The next lemma says that this property is shared hy *selR*.

Lemma *Suppose that R is well-bounded and transitive. Then*

$$\bigcup_{EA} \,;\, sel\,R \;=\; \exists\, sel\,R \,;\, sel\,R \;.$$

Proof

$$\exists\, sel\,R \,;\, sel\,R$$

$$= \quad \{\ \text{def. selector of } R \text{ (eq. 8.8)}\ \}$$

$$\exists\, sel\,R \,;\, \exists\, \eta_A^\circ \,;\, \widehat{max}\,R$$

$$= \quad \{\ \exists \text{ functor}\ \}$$

$$\exists(sel\,R \,;\, \eta_A^\circ) \,;\, \widehat{max}\,R$$

$$= \quad \{\ \text{def. selector of } R\ \}$$

$$\exists(\exists\, \eta_A^\circ \,;\, \widehat{max}\,R \,;\, \eta_A^\circ) \,;\, \widehat{max}\,R$$

$$= \quad \{\ \text{relation totaliser}\ \}$$

$$\exists(\exists\, \eta_A^\circ \,;\, max\,R) \,;\, \widehat{max}\,R$$

$$= \quad \{\ \exists \text{ functor}\ \}$$

$$\exists\, \exists\, \eta_A^\circ \,;\, \exists\, max\,R \,;\, \widehat{max}\,R$$

$$= \quad \{\ \exists\, max\,R \text{ map, relation totaliser (prop. 7.3.2)}\ \}$$

$$\exists\, \exists\, \eta_A^\circ \,;\, (\exists\, max\,R \,;\, max\,R)^{\smile}$$

$$= \quad \{\ \text{theorem 8.1.7}\ \}$$

$$\exists\, \exists\, \eta_A^\circ \,;\, (\bigcup_A \,;\, max\,R)^{\smile}$$

$$= \quad \{\ \bigcup_A \text{ map, relation totaliser (prop. 7.3.2)}\ \}$$

$$\exists\, \exists\, \eta_A^\circ \,;\, \bigcup_A \,;\, \widehat{max}\,R$$

$$= \quad \{\ \bigcup : \exists \circ \exists \rightarrow \exists\ \}$$

$$\bigcup_{EA} \,;\, \exists\, \eta_A^\circ \,;\, \widehat{max}\,R$$

$=$ { def. selector of R }

$$\bigcup_{EA} ; selR$$

8.2.3.3 The next lemma is a technical result that we shall need below. It makes use of the Smyth and Hoare extensions of a relation $R : A \rightarrow B$, which were introduced in paragraph 6.5.6.

Lemma *Let $T : A \rightarrow B$. Then*

$$(\exists T)^\circ ; \exists^{*}\tilde{T} ; \exists \eta_B^\circ \ \subset \ \sqsupseteq_B \cap \mathrm{S}(T^\circ ; T) \ .$$

Proof

$$(\exists T)^\circ ; \exists^{*}\tilde{T} ; \exists \eta_B^\circ$$

$\subset$ { proposition 6.5.7 }

$$(\exists T)^\circ ; \mathrm{H}\tilde{T} ; \exists \eta_B^\circ$$

$=$ { Hoare extension (eq. 6.28) }

$$(\exists T)^\circ ; \exists \tilde{T} ; \sqsupseteq_{EB} ; \exists \eta_B^\circ$$

$\subset$ { weak naturality of $\sqsupseteq$ (lemma 6.5.5.1) }

$$(\exists T)^\circ ; \exists \tilde{T} ; \exists \eta_B^\circ ; \sqsupseteq_B$$

$=$ { $\exists$ functor }

$$(\exists T)^\circ ; \exists(\tilde{T} ; \eta_B^\circ) ; \sqsupseteq_B$$

$=$ { relation totaliser }

$$(\exists T)^\circ ; \exists T ; \sqsupseteq_B$$

$\subset$ { $\exists T$ simple }

$$\sqsupseteq_B$$

It now remains to show that

$$(\exists T)^\circ ; \exists^{*}\tilde{T} ; \exists \eta_B^\circ \ \subset \ \mathrm{S}(T^\circ ; T) \ ,$$

or equivalently (by definition of the Smyth extension)

$$\in_B ; (\exists T)^\circ ; \exists^{*}\tilde{T} ; \exists \eta_B^\circ \ \subset \ T^\circ ; T ; \in_B \ .$$

Here is a proof of the latter containment:

$$
\begin{array}{ll}
 & \in_B ; (\exists T)^\circ ; \exists^\bullet \tilde{T} ; \exists \eta_B^\circ \\
= & \quad \{ \text{ reciprocal, } \ni : \exists \rightarrow \mathit{Rel}(\mathcal{E}) \ \} \\
 & T^\circ ; \in_A ; \exists^\bullet \tilde{T} ; \exists \eta_B^\circ \\
\subset & \quad \{ \text{ proposition 6.5.7 } \} \\
 & T^\circ ; \in_A ; S\tilde{T} ; \exists \eta_B^\circ \\
\subset & \quad \{ \text{ def. Smyth extension } \} \\
 & T^\circ ; \tilde{T} ; \in_{EB} ; \exists \eta_B^\circ \\
= & \quad \{ \text{ domain calculus (prop. 5.4.6) } \} \\
 & T^\circ ; {}^<T ; \tilde{T} ; \in_{EB} ; \exists \eta_B^\circ \\
\subset & \quad \{ \text{ relation totaliser } \} \\
 & T^\circ ; T ; \eta_B ; \in_{EB} ; \exists \eta_B^\circ \\
= & \quad \{ \text{ reciprocal, } \ni : \exists \rightarrow \mathit{Rel}(\mathcal{E}) \ \} \\
 & T^\circ ; T ; \in_B ; (\exists \eta_B^\circ)^\circ ; \exists \eta_B^\circ \\
\subset & \quad \{ \ \exists \eta_B^\circ \text{ simple } \} \\
 & T^\circ ; T ; \in_B
\end{array}
$$

8.2.3.4 Let x be a subset of A. Suppose that you have a subset y of x such that for each a in x there exists an element b in y which is above a:

$$\forall a \in x : \exists b \in y : a(R)b \ .$$

Then a maximum element of y is also a maximum element of x. The lemma below is a formal statement of this fact.

Lemma *If $R : A \rightarrow A$ is transitive,*

$$(\sqsupseteq_A \cap SR) ; \mathit{max}R \ \subset \ \mathit{max}R \ .$$

Proof By definition of $\mathit{max}R$, it suffices to show

$$(\sqsupseteq_A \cap SR) ; \mathit{max}R \ \subset \ \ni_A$$

and

$$(\sqsupseteq_A \cap SR)\,;\, max\, R \;\subset\; R{\uparrow} \,.$$

Here is a proof of the first containment:

$$\begin{array}{ll}
 & (\sqsupseteq_A \cap SR)\,;\, max\, R \\
\subset & \quad \{\ \text{monotonicity}\ \} \\
 & \sqsupseteq_A \,;\, max\, R \\
= & \quad \{\ \text{def. maximum (eq. 8.5)}\ \} \\
 & \sqsupseteq_A \,;\, (R{\uparrow} \cap \ni_A) \\
\subset & \quad \{\ \text{monotonicity}\ \} \\
 & \sqsupseteq_A \,;\, \ni_A \\
= & \quad \{\ \text{division calculus}\ \} \\
 & \ni_A
\end{array}$$

The other containment,

$$(\sqsupseteq_A \cap SR)\,;\, max\, R \;\subset\; R{\uparrow}$$

is equivalent to

$$\in_A \,;\, (\sqsupseteq_A \cap SR)\,;\, max\, R \;\subset\; R \,,$$

by definition of (-)↑ and left-division.

$$\begin{array}{ll}
 & \in_A \,;\, (\sqsupseteq_A \cap SR)\,;\, max\, R \\
\subset & \quad \{\ \text{monotonicity}\ \} \\
 & \in_A \,;\, SR \,;\, max\, R \\
\subset & \quad \{\ \text{def. Smyth extension (eq. 6.25)}\ \} \\
 & R \,;\, \in_A \,;\, max\, R \\
\subset & \quad \{\ \text{def. maximum}\ \} \\
 & R \,;\, \in_A \,;\, R{\uparrow} \\
\subset & \quad \{\ \text{def. upper bound}\ \} \\
 & R \,;\, R \\
\subset & \quad \{\ R \text{ transitive}\ \} \\
 & R
\end{array}$$

8.2.3.5 The next lemma completes the proof that $selR$ is a selector. It is worthwhile to observe that unlike the preceding lemmas, this result makes use of all properties of R: reflexivity, transitivity, and well–boundedness.

Lemma *Suppose that $R : A \to A$ is a well–bounded preorder. Then*

$$\exists^{\star} selR \,;\, selR \;\subset\; \exists\, selR \,;\, selR \,.$$

Proof

$$
\begin{array}{ll}
 & \exists^{\star} selR \,;\, selR \;\subset\; \exists\, selR \,;\, selR \\
\Leftrightarrow & \quad \{ \text{ def. selector of } R \text{ (eq. 8.8) } \} \\
 & \exists^{\star} selR \,;\, \exists\, \eta_A^{\circ} \,;\, \widetilde{max} R \;\subset\; \exists\, selR \,;\, \exists\, \eta_A^{\circ} \,;\, \widetilde{max} R \\
\Leftrightarrow & \quad \{ \text{ claim (8.9) below, } \exists\, selR \text{ and } \exists\, \eta_A^{\circ} \text{ maps, prop. 7.3.2 } \} \\
 & (\exists^{\star} selR \,;\, \exists\, \eta_A^{\circ} \,;\, max R)^{\sim} \;\subset\; (\exists\, selR \,;\, \exists\, \eta_A^{\circ} \,;\, max R)^{\sim} \\
\Leftrightarrow & \quad \{ \text{ claim (8.10) below, prop. 7.2.5 } \} \\
 & \exists^{\star} selR \,;\, \exists\, \eta_A^{\circ} \,;\, max R \;\subset\; \exists\, selR \,;\, \exists\, \eta_A^{\circ} \,;\, max R \\
\Leftrightarrow & \quad \{ \text{ def. selector of } R \text{ (eq. 8.8), } \exists \text{ functor, relation totaliser } \} \\
 & \exists\, \exists\, \eta_A^{\circ} \,;\, \exists^{\star} \widetilde{max} R \,;\, \exists\, \eta_A^{\circ} \,;\, max R \;\subset\; \exists\, \exists\, \eta_A^{\circ} \,;\, \exists\, max R \,;\, max R \\
\Leftrightarrow & \quad \{ \; \eta_A \text{ monic } \} \\
 & \exists^{\star} \widetilde{max} R \,;\, \exists\, \eta_A^{\circ} \,;\, max R \;\subset\; \exists\, max R \,;\, max R \\
\Leftrightarrow & \quad \{ \; \exists\, max R \text{ map } \} \\
 & (\exists\, max R)^{\circ} \,;\, \exists^{\star} \widetilde{max} R \,;\, \exists\, \eta_A^{\circ} \,;\, max R \;\subset\; max R \\
\Leftarrow & \quad \{ \text{ lemma 8.2.3.3 } \} \\
 & (\sqsupseteq_A \cap\, \mathrm{S}(max^{\circ} R \,;\, max R)) \;;\; max R \;\subset\; max R \\
\Leftarrow & \quad \{ \text{ Smyth extension is monotonic, prop. 8.1.4 } \} \\
 & (\sqsupseteq_A \cap\, \mathrm{S} R) \;;\; max R \;\subset\; max R \\
\Leftrightarrow & \quad \{ \text{ lemma 8.2.3.4 } \} \\
 & \text{true}
\end{array}
$$

There are still two claims to be checked:

$$^{<}((\exists^{\star} selR)^{\circ} \,;\, \exists^{\star} selR \,;\, \exists\, \eta_A^{\circ} \,;\, max R) \;\subset\; ^{<}(\exists\, \eta_A^{\circ} \,;\, max R) \,, \qquad (8.9)$$

and

$$ {}^{<}(\exists^\star sel R \,;\, \exists\,\eta_A^\circ \,;\, max\, R) \;=\; {}^{<}(\exists\, sel R \,;\, \exists\,\eta_A^\circ \,;\, max\, R)\ . \qquad (8.10)$$

We shall only prove claim (8.9); the proof of (8.10) is similar.

$$\begin{array}{ll}
 & {}^{<}((\exists^\star sel R)^\circ \,;\, \exists^\star sel R \,;\, \exists\,\eta_A^\circ \,;\, max\, R) \\
= & \quad \{\ R \text{ well-bounded, domain calculus (5.4.6)}\ \} \\
 & {}^{<}((\exists^\star sel R)^\circ \,;\, \exists^\star sel R \,;\, \exists\,\eta_A^\circ \,;\, \ni_A) \\
= & \quad \{\ \ni : \exists \rightarrow Rel(\mathcal{E})\ \} \\
 & {}^{<}((\exists^\star sel R)^\circ \,;\, \exists^\star sel R \,;\, \ni_{EA} \,;\, \eta_A^\circ) \\
= & \quad \{\ \exists^\star \text{ reciprocal-preserving functor}\ \} \\
 & {}^{<}(\exists^\star(sel^\circ R \,;\, sel R) \,;\, \ni_{EA} \,;\, \eta_A^\circ) \\
= & \quad \{\ (sel^\circ R \,;\, sel R) \text{ entire by 8.2.3.1, proposition 6.5.3}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, sel^\circ R \,;\, sel R \,;\, \eta_A^\circ) \\
= & \quad \{\ \text{def. selector of } R\text{, relation totaliser}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, (\widetilde{max}\, R)^\circ \,;\, (\exists\,\eta_A^\circ)^\circ \,;\, \exists\,\eta_A^\circ \,;\, max\, R) \\
\subset & \quad \{\ \exists\,\eta_A^\circ \text{ simple}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, (\widetilde{max}\, R)^\circ \,;\, max\, R) \\
= & \quad \{\ \text{domain calculus}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, (\widetilde{max}\, R)^\circ \,;\, {}^{<}(max\, R)) \\
= & \quad \{\ \text{coreflexive implies symmetric}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, ({}^{<}(max\, R) \,;\, \widetilde{max}\, R)^\circ) \\
= & \quad \{\ \text{relation totaliser}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, (max\, R \,;\, \eta_A)^\circ) \\
= & \quad \{\ \text{reciprocal}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, \eta_A^\circ \,;\, max^\circ R) \\
\subset & \quad \{\ \text{domain calculus}\ \} \\
 & {}^{<}(\ni_{EA} \,;\, \eta_A^\circ) \\
= & \quad \{\ \ni : \exists \rightarrow Rel(\mathcal{E})\ \} \\
 & {}^{<}(\exists\,\eta_A^\circ \,;\, \ni_A) \\
= & \quad \{\ R \text{ well-bounded, domain calculus}\ \} \\
 & {}^{<}(\exists\,\eta_A^\circ \,;\, max\, R)
\end{array}$$

This completes the proof of the lemma, and therefore the proof of theorem 8.2.3.

8.3 Monotonicity implies Distributivity

The function that returns the minimum of a set of natural numbers can be defined as the selector of $\geq$. Indeed, since $\geq$ is a partial order, $max(\geq)$ is a simple relation. Because the relation totaliser is also a simple–relation classifier, $\widetilde{max}(\geq)$ is a total function, and therefore $sel(\geq)$ is a total function as well. In fact, $sel(\geq)$ is just the function $\sqcap$ that returns the minimum of a set of natural numbers; the minimum of the empty set is the fictitious value infinity. The function $\sqcap$ distributes over addition in the following sense:

$$\sqcap\{a+b \mid a \in x, b \in y\} \;=\; \sqcap x \,+\, \sqcap y \;.$$

This distributivity property is satisfied because addition is monotonic with respect to $\geq$. In this section, we shall investigate in what sense monotonicity implies distributivity. As before, we start with maximum elements $(max\,R)$, and we consider $(sel\,R)$ later.

8.3.1 Proposition *Let $\mathcal{E}$ be a topos, and let F be a polynomial endofunctor on $\mathcal{E}$. Let $h : FA \to A$ be a functional F–algebra, and let R be an endorelation on A. Finally, let $\boxed{?}$ be $(\subseteq)$, $(\supseteq)$ or $(=)$. If h is monotonic with respect to R,*

$$\begin{tikzcd}
A \arrow[d, dash, "R"'] \arrow[dr, phantom, "\boxed{?}"] & FA \arrow[l, "h"'] \arrow[d, "F^{*}R"] \\
A \arrow[r, "h^{\circ}"'] & FA
\end{tikzcd}$$

then h distributes over $(max\,R)$:

$$\begin{tikzcd}
PFA \arrow[d, "\exists h"'] & FPA \arrow[l, "F\dagger_A"'] \arrow[r, "F^{*}max\,R"] \arrow[d, phantom, "\boxed{?}"] & FA \arrow[d, dash, "FA"] \\
PA \arrow[r, "max\,R"'] & A & FA \arrow[l, "h"]
\end{tikzcd}$$

Proof The proof is presented in a top-down fashion: here we give the main calculation, and the two lemmas that are used in this calculation are proved later.

$$\begin{array}{ll}
 & F\dagger_A ; \exists\, h ; \mathit{max}R \\
= & \quad \{ \text{ lemma 8.3.1.1 below } \} \\
 & (F^{\star}\in_A)\backslash(h ; R) \cap (F^{\star}\ni_A ; h) \\
= & \quad \{ \text{ modular identity } \} \\
 & (((F^{\star}\in_A)\backslash(h ; R) \; ; \; h^{\circ}) \cap F^{\star}\ni_A) \; ; \; h \\
= & \quad \{ \text{ division calculus (prop. 6.4.2) } \} \\
 & ((F^{\star}\in_A)\backslash(h ; R ; h^{\circ}) \cap F^{\star}\ni_A) \; ; \; h \\
\boxed{?} & \quad \{ \text{ assumption, } (F^{\star}\in_A)\backslash(\text{-}) \text{ monotonic } \} \\
 & ((F^{\star}\in_A)\backslash F^{\star}R \cap F^{\star}\ni_A) \; ; \; h \\
= & \quad \{ \text{ lemma 8.3.1.2 below, def. maximnm } \} \\
 & F^{\star}\mathit{max}R ; h
\end{array}$$

8.3.1.1 The next lemma shows how the composite $F\dagger_A ; \exists\, h ; \mathit{max}R$ can be expressed in terms of more primitive operators.

Lemma

$$F\dagger_A ; \exists\, h ; \mathit{max}R \;=\; (F^{\star}\in_A)\backslash(h ; R) \cap (F^{\star}\ni_A ; h)$$

Proof

$$\begin{array}{ll}
 & F\dagger_A ; \exists\, h ; \mathit{max}R \\
= & \quad \{ \; F\dagger = \Lambda F^{\star}\ni, \text{ power adjunction } \} \\
 & \Lambda(F^{\star}\ni_A ; h) ; \mathit{max}R \\
= & \quad \{ \text{ maximum (eq. 8.6) } \} \\
 & (F^{\star}\ni_A ; h)^{\circ}\backslash R \cap (F^{\star}\ni_A ; h) \\
= & \quad \{ \text{ division calculus (prop. 6.4.2) } \} \\
 & (F^{\star}\in_A)\backslash(h ; R) \cap (F^{\star}\ni_A ; h)
\end{array}$$

8.3.1.2 In section 6.4.7, it was shown that polynomial functors preserve strict left-division. The original motivation to prove that theorem was the following result, whicb shows how polynomial functors can be distributed over the definition of maximum elements.

Lemma *Let T and S be relations with the same source and target. If F is a polynomial functor,*

$$F^\star(T\backslash S \cap T^\circ) \;=\; F^\star T\backslash F^\star S \cap F^\star T^\circ \ .$$

Proof

$$\begin{array}{ll}
 & F^\star(T\backslash S \cap T^\circ) \\
= & \quad \{\ F \text{ polynomial, props. 6.3.10 and 5.3.9 }\ \} \\
 & F^\star(T\backslash S) \cap F^\star T^\circ \\
= & \quad \{\ \text{see below}\ \} \\
 & F^\star T\backslash F^\star S \cap F^\star T^\circ
\end{array}$$

To prove inclusion ($\subset$) in the last step, one may reason as follows

$$\begin{array}{ll}
 & F^\star(T\backslash S) \subset (F^\star T)\backslash(F^\star S) \\
\Leftrightarrow & \quad \{\ \text{left-division}\ \} \\
 & F^\star T \,;\, F^\star(T\backslash S) \subset F^\star S \\
\Leftrightarrow & \quad \{\ F^\star \text{ functor}\ \} \\
 & F^\star(T \,;\, (T\backslash S)) \subset F^\star S \\
\Leftrightarrow & \quad \{\ \text{left-division, } F^\star \text{ monotonic}\ \} \\
 & \text{true}
\end{array}$$

The converse inclusion,

$$F^\star T\backslash F^\star S \cap F^\star T^\circ \subset F^\star(T\backslash S) \ ,$$

is proved below:

$$F^\star T\backslash F^\star S \cap F^\star T^\circ$$

$\subset$ { domain calculus }

$<(F^\star T^\circ) \,;\, F^\star T \backslash F^\star S$

$=$ { def. strict left-division (par. 6.4.6) }

$F^\star T \backslash\backslash\, F^\star S$

$=$ { F polynomial, prop. 6.4.7 }

$F^\star(T \backslash\backslash\, S)$

$\subset$ { def. strict left-division, $F^\star$ monotonic }

$F^\star(T \backslash S)$

8.3.2 In the preceding proposition, it was established that monotonicity implies distributivity for the maximum relation of R. Not surprisingly, this result carries over to the selector of R. There are two significant differences, however. The first is that the topos under consideration is Boolean — this is to ensure the existence of a relation totaliser. The second difference is that the relation R should be well-bonnded. This condition is used in the proof given below; I am not sure whether it could be eliminated.

Theorem *Let $\mathcal{E}$ be a Boolean topos, and let F be a polynomial endofunctor on $\mathcal{E}$. Let $h : FA \rightarrow A$ be a functional F-algebra, and let R be a well-bounded endorelation on A. Finally, let $\boxed{?}$ be $(\subseteq)$, $(\supseteq)$ or $(=)$. If h is monotonic with respect to R,*

$$\begin{tikzcd}
A \arrow[d, "R"'] \arrow[rd, phantom, "\boxed{?}"] & FA \arrow[l, "h"'] \arrow[d, "F^\star R"] \\
A \arrow[r, "h^\circ"'] & FA
\end{tikzcd}$$

then h distributes over $(sel R)$:

$$\begin{tikzcd}
PFEA \arrow[d, "\exists(\overline{F}_A \,;\, Eh)"'] & FPEA \arrow[l, "F\dagger_{EA}"'] \arrow[r, "F^\star sel R"] \arrow[d, phantom, "\boxed{?}"] & FEA \arrow[d, "\overline{F}_A"] \\
PEA \arrow[r, "sel R"'] & EA & EFA \arrow[l, "Eh"]
\end{tikzcd}$$

Proof The proof is split into three lemmas. The main argument is given below, and then the lemmas are proved in detail.

$$\begin{array}{ll}
 & F\dagger_{EA}\,;\exists(\overline{F}_A\,;Eh)\,;\mathit{sel}R\ \boxed{?}\ F^{\star}\mathit{sel}R\,;\overline{F}_A\,;Eh\\
\Leftrightarrow & \quad \{\text{ lemmas 8.3.2.1 and 8.3.2.2 below }\}\\
 & F\,\exists\,\eta^{\circ}_A\,;(F\dagger_A\,;\exists\,h\,;\mathit{max}R)^{\sim}\ \boxed{?}\ F\,\exists\,\eta^{\circ}_A\,;(F^{\star}\mathit{max}\,R\,;h)^{\sim}\\
\Leftrightarrow & \quad \{\ \eta \text{ monic }\}\\
 & (F\dagger_A\,;\exists\,h\,;\mathit{max}R)^{\sim}\ \boxed{?}\ (F^{\star}\mathit{max}R\,;h)^{\sim}\\
\Leftrightarrow & \quad \{\text{ prop. 7.2.5, lemma 8.3.2.3 below }\}\\
 & F\dagger_A\,;\exists\,h\,;\mathit{max}R\ \boxed{?}\ F^{\star}\mathit{max}R\,;h\\
\Leftarrow & \quad \{\text{ prop. 8.3.1 }\}\\
 & h\,;R\,;h^{\circ}\ \boxed{?}\ F^{\star}R
\end{array}$$

8.3.2.1 In the first lemma, we aim to factor the operators of the relation totaliser (η and $(\text{-})^{\sim}$) to the outside of the expression $F\dagger_{EA}\,;\exists(\overline{F}_A\,;Eh)\,;\mathit{sel}R$. This goal is motivated by a wish to eliminate these operators from our proof obligations altogether.

Lemma

$$F\dagger_{EA}\,;\exists(\overline{F}_A\,;Eh)\,;\mathit{sel}R \;=\; F\,\exists\,\eta^{\circ}_A\,;(F\dagger_A\,;\exists\,h\,;\mathit{max}\,R)^{\sim}$$

Proof

$$\begin{array}{ll}
 & F\dagger_{EA}\,;\exists(\overline{F}_A\,;Eh)\,;\mathit{sel}R\\
= & \quad \{\text{ def. selector of } R\ \}\\
 & F\dagger_{EA}\,;\exists(\overline{F}_A\,;Eh)\,;\exists\,\eta^{\circ}_A\,;\widetilde{\mathit{max}}R\\
= & \quad \{\ \exists \text{ functor }\}\\
 & F\dagger_{EA}\,;\exists(\overline{F}_A\,;Eh\,;\eta^{\circ}_A)\,;\widetilde{\mathit{max}}R\\
= & \quad \{\text{ def. } \overline{F}\text{, relation totaliser }\}\\
 & F\dagger_{EA}\,;\exists(F^{\star}\eta^{\circ}_A\,;h)\,;\widetilde{\mathit{max}}R
\end{array}$$

$$
\begin{array}{ll}
= & \{\ \exists \text{ functor }\ \} \\
& F\dagger_{EA} ; \exists F^{\star}\eta^{\circ}_{A} ; \exists h ; \widetilde{max}\, R \\
= & \{\ F\dagger : F \circ \exists \rightarrow \exists \circ F^{\star} \text{ (prop. 6.2.8) }\ \} \\
& F \exists \eta^{\circ}_{A} ; F\dagger_{A} ; \exists h ; \widetilde{max}\, R \\
= & \{\ (F\dagger_{A} ; \exists h) \text{ map, relation totaliser (prop. 7.3.2) }\ \} \\
& F \exists \eta^{\circ}_{A} ; (F\dagger_{A} ; \exists h ; max\, R)^{\sim}
\end{array}
$$

8.3.2.2 The next lemma is motivated by the same observation as the preceding result. Here the goal is to factor the operators of the relation totaliser to the outside of the expression $F^{\star} sel\, R ; \overline{F}_{A} ; Eh$.

Lemma

$$
F^{\star} sel\, R ; \overline{F}_{A} ; Eh \quad = \quad F \exists \eta^{\circ}_{A} ; (F^{\star} max\, R ; h)^{\sim}
$$

Proof

$$
\begin{array}{ll}
& F^{\star} sel\, R ; \overline{F}_{A} ; Eh \\
= & \{\ \text{def. selector of } R\ \} \\
& F^{\star}(\exists \eta^{\circ}_{A} ; \widetilde{max}\, R) ; \overline{F}_{A} ; Eh \\
= & \{\ F \text{ relator, } \exists \eta^{\circ}_{A} \text{ map }\ \} \\
& F \exists \eta^{\circ}_{A} ; F^{\star}\widetilde{max}\, R ; \overline{F}_{A} ; Eh \\
= & \{\ \text{prop. 7.3.4 }\ \} \\
& F \exists \eta^{\circ}_{A} ; (F^{\star} max\, R)^{\sim} ; Eh \\
= & \{\ \text{prop. 7.3.3 }\ \} \\
& F \exists \eta^{\circ}_{A} ; (F^{\star} max\, R ; h)^{\sim}
\end{array}
$$

8.3.2.3 To prove the equivalence

$$
\begin{array}{ll}
& (F\dagger_{A} ; \exists h ; max\, R)^{\sim} \ \boxed{?}\ (F^{\star} max\, R ; h)^{\sim} \\
\Leftrightarrow & \\
& F\dagger_{A} ; \exists h ; max\, R \ \boxed{?}\ F^{\star} max\, R ; h
\end{array}
$$

it suffices (hy prop. 7.2.5) to show that both sides in the latter inequation have the same domain. This is done in the following lemma.

Lemma *Suppose that R is well-bounded. Then*

$$ {}^{<}(F\dagger_A \,;\, \exists h \,;\, \mathit{max}R) \;=\; {}^{<}(F^{\star}\mathit{max}R \,;\, h)\ . $$

Proof

$$
\begin{array}{cl}
 & {}^{<}(F\dagger_A \,;\, \exists h \,;\, \mathit{max}R) \\
= & \quad \{\ F\dagger = \Lambda F^{\star}\ni,\ \text{power adjunction}\ \} \\
 & {}^{<}(\Lambda(F^{\star}\ni_A \,;\, h) \,;\, \mathit{max}R) \\
= & \quad \{\ R\ \text{well-bounded, domain calculus (prop. 5.4.6)}\ \} \\
 & {}^{<}(\Lambda(F^{\star}\ni_A \,;\, h) \,;\, \ni_A) \\
= & \quad \{\ \text{power adjunction}\ \} \\
 & {}^{<}(F^{\star}\ni_A \,;\, h) \\
= & \quad \{\ h\ \text{entire, domain calculus}\ \} \\
 & {}^{<}(F^{\star}\ni_A) \\
= & \quad \{\ \text{relators preserve domain (prop. 5.4.7)}\ \} \\
 & F^{\star}\,{}^{<}\!\ni_A \\
= & \quad \{\ R\ \text{well-bounded}\ \} \\
 & F^{\star}\,{}^{<}(\mathit{max}R) \\
= & \quad \{\ \text{relators preserve domain}\ \} \\
 & {}^{<}(F^{\star}\mathit{max}R) \\
= & \quad \{\ h\ \text{entire, domain calculus}\ \} \\
 & {}^{<}(F^{\star}\mathit{max}R \,;\, h)
\end{array}
$$

8.3.3 The next proposition is often helpful in checking the applicability conditions of the two preceding results. It is an immediate consequence of the definition of initial F-algebras.

Proposition *Let $\mathcal{E}$ be a regular category, and let $F : \mathcal{E} \rightarrow \mathcal{E}$ be a relator that bas initial algebra $\mu(F)$. Furthermore, let $h : FA \rightarrow A$ be an F-algebra. Let $\boxed{?}$ be $(\subseteq)$, $(\supseteq)$ or $(=)$. Then*

$$\begin{tikzcd}
A \arrow[d, "R"'] \arrow[dr, phantom, "\boxed{?}"] & FA \arrow[l, "h"'] \arrow[d, "F^{\star}R"] \\
A \arrow[r, "h^{\circ}"'] & FA
\end{tikzcd}$$

implies

$$\begin{tikzcd}
T \arrow[d, "S"'] \arrow[dr, phantom, "\boxed{?}"] & FT \arrow[l, "\mu(F)"'] \arrow[d, "F^{\star}S"] \\
T \arrow[r, "(\mu(F))^{\circ}"'] & FT
\end{tikzcd}$$

where $S = \llparenthesis h \rrparenthesis \; ; R \; ; \llparenthesis h \rrparenthesis^{\circ}$.

8.3.4 Is it worthwhile to try and generalise the results of this chapter to arbitrary toposes, perhaps by other means than relation totalisers? Typical applications of the theorems presented here involve the notion of natural numbers: a shortest sequence, a tree of minimum height, a coarsest partition. In all these applications, one makes use of the standard partial order on natural numbers, which is defined by

$$(\geq) \quad = \quad (+)^{\circ} \, ; \pi_2 \, .$$

In his thesis about *Order and Recursion in Topoi*, Brook has proved that $\geq$ is well-bounded iff the topos under consideration is Boolean ([19], p. 156). It is unlikely, therefore, that any interesting theory about optimisation operators can be developed in toposes which are not Boolean.

9 Dynamic Programming

In the computing literature, dynamic programming is often described as a class of algorithms rather than of specifications. However, as most computing scientists would agree, the specifications of dynamic programming algoritbms constitute a class of problems that are essentially tbe same. We propose a precise definition of this problem class by phrasing a generic specification in categorical terms. Using the calculus of relations developed in the preceding chapters, it is shown how one may derive an abstract algorithm from this specification.

The result is illustrated by considering a particular example (text formatting) in some detail. It will be shown how the hasic theorem abont dynamic programming can be applied by mechanically instantiating the abstract definitions. More examples of dynamic programming can be found in the first part of this thesis.

Although the basic theorem about dynamic programming is satisfactory from a theoretical point of view, it is difficult to apply in practice. Motivated by this observation, we simplify the basic theorem, using our earlier results about relation totalisers. There is a small price to pay for the more practical result: we have to assume that the topos under consideration is Boolean.

9.1 The Basic Theorem

9.1.1 Throughout this section, it is assumed that we are working in a topos $\mathcal{E}$. Let $R : A \rightarrow B$ be a relation in $Rel(\mathcal{E})$. The *inverse image function* of R is the power transpose of its converse:

$$InvR \;=\; \Lambda R^{\circ} : B \rightarrow PA \ .$$

In *Set*, the inverse image function of R may be characterised by the following set comprehension:

$$(InvR)b \;=\; \{\, a \mid a(R)b \,\} \ .$$

9.1.2 Let $F : \mathcal{E} \rightarrow \mathcal{E}$ be a relator that has initial F–algebra $\mu(F) : FT \rightarrow T$. Furthermore, let $S : FB \rightarrow B$ be a relational $F^{\star}$–algebra, and let $h : FA \rightarrow A$ be an F–algebra. Finally, let $M : PA \rightarrow A$ be a relation. We shall study a relation K in $Rel(\mathcal{E})$, defined by

$$K \;=\; Inv(\![S]\!)_{F^{\star}} \,;\, \exists\,(\![h]\!)_{F^{\star}} \,;\, M \ .$$

We shall refer to this definition as the *generic specification*, because many dynamic programming problems can be cast into this form.

9.1.3 It seems premature to treat instances of the generic specification in full detail now; we confine ourselves to giving an intuitive interpretation in $Rel(Set)$. Full examples, along with concrete algorithms, will be discussed after we have derived an abstract algorithm from the generic specification.

Tbe inverse homomorphism ($Inv(\llparenthesis S \rrparenthesis)$) is typically used to generate a set of comhinatorial objects, like all permutations of a bag or all partitions of an integer. The latter example is obtained by taking for $\llparenthesis S \rrparenthesis$ the function *sum* that sums the elemeuts of a list of positive integers. Given an integer ($n > 0$), the expression $(Inv(sum))\, n$ stands for the set of all lists whose elements add up to n.

Usually, one does uot care about the order of the elements in an integer partition, aud therefore it is better to consider an integer partition as a bag rather than a list. This is a typical function of the existential image ($\exists\, \llparenthesis h \rrparenthesis$); it turns every list generated by the inverse homomorphism ($Inv(\llparenthesis S \rrparenthesis)$) into a bag.

Now suppose all partitions are assigned a cost, and we are interested in finding a *best* partition of minimum cost. The selection of such an optimal element is modelled by the relation M; it picks an element of minimum cost, possibly in a nondeterministic fashion.

The goal is now to derive an abstract algorithm from the generic specification

$$K \;=\; Inv(\llparenthesis S \rrparenthesis)\,;\,\exists\,\llparenthesis h \rrparenthesis\,;\,M\ .$$

The proofs that are involved in this derivation will be given in detail, to demonstrate how our calculus facilitates straightforward, almost mechanical proofs.

9.1.4 Lemma (Goguen [43]) *Folds can be characterised as follows:*

$$(T = \llparenthesis S \rrparenthesis) \;\Leftrightarrow\; (T = (\mu(F))^\circ\,;\,F^\star T\,;\,S)\ .$$

Proof

$$\begin{array}{ll} & T = \llparenthesis S \rrparenthesis \\ \Leftrightarrow & \quad \{\ \text{initiality of } \mu(F)\ \} \\ & \mu(F)\,;T \;=\; F^\star T\,;S \end{array}$$

$$
\begin{array}{ll}
\Leftrightarrow & \{\ \mu(F) \text{ isomorphism (prop. 6.3.6) }\} \\
 & T \ = \ (\mu(F))^{\circ} \,;\, F^{\star}T \,;\, S
\end{array}
$$

9.1.5 Lemma *Let t be the composite*

$$t \ = \ \mathit{Inv}\llparenthesis S \rrparenthesis \,;\, \exists\, \llparenthesis h \rrparenthesis \ .$$

Then t satisfies the following equation:

$$t \ = \ \mathit{Inv}\, S \ ; \ \exists\,(F t \,;\, F\dagger_A \,;\, \exists\, h) \ ; \ \bigcup_A \ .$$

Proof

$$
\begin{array}{ll}
 & t \\
= & \{\ \text{def. } t\ \} \\
 & \mathit{Inv}\llparenthesis S \rrparenthesis \,;\, \exists\, \llparenthesis h \rrparenthesis \\
= & \{\ \text{power adjunction}\ \} \\
 & \Lambda\,(\llparenthesis S \rrparenthesis^{\circ} \,;\, \llparenthesis h \rrparenthesis) \\
= & \{\ \text{lemma 9.1.4}\ \} \\
 & \Lambda\,(\,((\mu(F))^{\circ} \ ; \ F^{\star}\llparenthesis S \rrparenthesis \ ; \ S)^{\circ} \ ; \ \llparenthesis h \rrparenthesis\,) \\
= & \{\ \text{reciprocal}\ \} \\
 & \Lambda\,(\,S^{\circ} \ ; \ F^{\star}\llparenthesis S \rrparenthesis^{\circ} \ ; \ \mu(F) \ ; \ \llparenthesis h \rrparenthesis\,) \\
= & \{\ F\text{-homomorphism}\ \} \\
 & \Lambda\,(\,S^{\circ} \ ; \ F^{\star}\llparenthesis S \rrparenthesis^{\circ} \ ; \ F\llparenthesis h \rrparenthesis \ ; \ h\,) \\
= & \{\ F \text{ relator}\ \} \\
 & \Lambda\,(\,S^{\circ} \ ; \ F^{\star}(\llparenthesis S \rrparenthesis^{\circ} \ ; \ \llparenthesis h \rrparenthesis) \ ; \ h\,) \\
= & \{\ \text{power adjunction, def. } t\ \} \\
 & \mathit{Inv}\, S \,;\, \exists\,(F t \,;\, F\dagger_A \,;\, \exists\, h) \,;\, \bigcup_A
\end{array}
$$

9.1.6 Theorem *Consider the generic specification*

$$K \;=\; Inv(\!|S|\!)_{F\star} \,;\, \exists\,(\!|h|\!)_{F\star} \,;\, M \;.$$

Suppose that

- *the relation $M : PA \rightharpoonup A$ is a selector, and*
- *the map h distributes over M in the following sense*

$$\begin{tikzcd}
FPA \arrow[rr, "F^{*}M"] \arrow[d, "F\dagger_A"'] & & FA \arrow[d, "h"] \\
PFA \arrow[r, "\exists h"'] \arrow[rru, phantom, "\boxed{?}"] & PA \arrow[r, "M"'] & A
\end{tikzcd}$$

where $\boxed{?}$ is $(\supseteq)$, $(\subseteq)$ or $(=)$.

Then K satisfies

$$\begin{tikzcd}
B \arrow[r, "Inv\,S"] \arrow[d, "K"'] \arrow[dr, phantom, "\boxed{?}"] & PFB \arrow[d, "\exists^{*}(F^{*}K\,;\,h)"] \\
A & PA \arrow[l, "M"]
\end{tikzcd}$$

Proof Let $t = Inv(\!|S|\!)\,;\exists\,(\!|h|\!)$, as in lemma 9.1.5. Then

$$\begin{array}{ll}
 & K \\
= & \{\ \text{defs. } K \text{ and } t\ \} \\
 & t\,;M \\
= & \{\ \text{lemma 9.1.5}\ \} \\
 & Inv\,S\,;\exists\,(Ft\,;F\dagger_A\,;\exists\,h)\,;\bigcup_A\,;M \\
= & \{\ M \text{ selector}\ \} \\
 & Inv\,S\,;\exists\,(Ft\,;F\dagger_A\,;\exists\,h)\,;\exists^{*}M\,;M \\
= & \{\ \exists \text{ relator}\ \} \\
 & Inv\,S\,;\exists^{*}(Ft\,;F\dagger_A\,;\exists\,h\,;M)\,;M
\end{array}$$

$$\begin{aligned} \boxed{?} \quad & \{\ h \text{ distributes over } M\ \} \\ & \mathit{Inv}\,S\,;\exists^{\star}(Ft\,;F^{\star}M\,;h)\,;M \\ = \quad & \{\ F \text{ relator, def. } K\ \} \\ & \mathit{Inv}\,S\,;\exists^{\star}(F^{\star}K\,;h)\,;M \end{aligned}$$

9.1.7 The preceding theorem is in line with traditional presentations of dynamic programming when the conclusion

$$K \ = \ \mathit{Inv}\,S\,;\exists^{\star}(F^{\star}K\,;h)\,;M$$

is given an operational interpretation. First, the argument is split in all possible ways by $(\mathit{Inv}\,S)$. The subproblems that have thus been generated are solved recursively with $(F^{\star}K)$, and the solutions to subproblems are composed into solutions for the whole problem by h. Finally, M selects an optimal element from this set of candidate solutions. This view of dynamic programming as a degenerate case of the divide–and–conquer strategy is voiced by Sedgewick in [82] and can be found in many other textbooks on algorithm design. Dynamic programming is degenerate in that one considers all possible splittings in $(\mathit{Inv}\,S)$ rather than just one element of $(\mathit{Inv}\,S)$, as one would do in a typical divide–and–conquer algorithm.

It should be stressed that the equation in theorem 9.1.6 is not necessarily an algorithm. It is always a valid equation, but it may be that $(\mathit{Inv}\,S)$ splits an argument x into x itself (and possibly something else), and so the recursion does not make progress. It is therefore necessary to verify termination whenever the theorem is used for program derivation. In most cases, this is a trivial exercise.

9.2 Application: Text Formatting

A well-known application of dynamic programming is the problem of breaking a sequence of words into lines to form a paragraph, such that the paragraph contains as little white space as possible. In this section, we shall not be concerned with constructing such an optimal layout itself; instead, we only consider the problem of computing the minimum amount of white space. In section 9.3.5 it will be shown how the same method applies to the construction of an optimal layout.

9.2.1 A first step towards a formal specification is to consider the types that are involved in this text formatting problem. The input sequence is represented by a list of natural numbers, each signifying the length of a word. The length of a word could be the number of characters it contains, but one might also chose a

`a concrete example that`	$[[1,8,7,4],$
`illustrates our representation`	$[11,3,14],$
`of paragraphs`	$[2,10]]$

Figure 9.1: An example paragraph

more complicated measure that depends on the width of individual characters. A paragraph is represented as a list whose elements are lists of natural numbers, and each of these component lists stands for a line in the paragraph. A concreteexample that illustrates this representation of paragraphs is given in figure 5.1. The data type of lists whose elements are lists of natural numbers was formally defined in section 6.3.1: it is the initial F–algebra, where $F : Set \rightarrow Set$ is the functor

$$Fk \;=\; L + (L \times k) \;,$$

and L is the set of lists of natural numbers.

We now aim to define the set of all possible layouts (paragraphs) for a given input sequence. The binary operator

$$+\!\!+ : (L \times L) \rightarrow L$$

concatenates two lists of numbers. Generalisiug this binary concatenation, the homomorphism

$$\llparenthesis [L, +\!\!+] \rrparenthesis$$

concatenates the components of a lists of lists; speaking in terms of the problem at hand, one could say that $\llparenthesis [L, +\!\!+] \rrparenthesis$ strings the lines of a paragraph together to form a single sequence of words. It follows that the inverse image of this operation, given by

$$Inv \llparenthesis [L, +\!\!+] \rrparenthesis$$

produces all possible layouts of its argument. For concreteness, here is an example:

$$\begin{aligned}(Inv \llparenthesis [L, +\!\!+] \rrparenthesis)\,[1,2,3] \;=\; \{\; & [[1,2,3]],\\ & [[1,2],\,[3]],\\ & [[1],\,[2,3]],\\ & [[1],\,[2],\,[3]]\;\}\;.\end{aligned}$$

Now that we know how to generate all possible layouts of a sequence of words, it remains to make the notion of *white space* more precise. Let $f : L \rightarrow (\mathrm{N} \cup \{\infty\})$ be a function that returns some measure of the amount of white space on a single line. To obtain nicely formatted paragraphs, one may have to chose a rather complicated function for f; fortunately, we do not need its formal definitioa here. Tbe function t returns a measure of the white space in the last line of a paragrapb, say l. If the length of l is below tbe maximum line length, $t(l)$ equals zero; otherwise, it is infinity. The total amount of wbite space in a paragraph is defined to be the sum of the white space in the component lines, as returned by the homomorphism

$$(\![[t,\otimes]]\!) \quad \text{where} \quad x \otimes n = f\,x + n \;.$$

The text formatting problem that we seek to solve can now be defined by the expression below:

$$\mathit{Inv}(\![[L,+\!\!+]]\!) \,;\, \exists\,(\![[t,\otimes]]\!) \,;\, \sqcap \;.$$

This very concise problem statement could be read as follows. First, the inverse homomorphism ($\mathit{Inv}(\![[L,+\!\!+]]\!)$) generates all possible layouts. Next, for each of these layouts, $(\![[t,\otimes]]\!)$ computes the amount of white space. This produces a set of natural numbers, and the function $\sqcap$ selects its minimum.

9.2.2 In order to derive an algorithm for tbe text formatting problem introduced above, we shall instantiate theorem 9.1.6. The instantiation is somewhat laborious, so before going through these manipulations in detail, we first discuss the final result. It makes use of a new notational convention: for auy binary operator $(\odot)$, $(\hat{\odot})$ defined by

$$(f \mathbin{\hat{\odot}} g)\,y \;=\; (f\,y) \odot (g\,y)$$

stands for its function–level counterpart.

Theorem *Let* $k \;=\; \mathit{Inv}(\![[L,+\!\!+]]\!) \,;\, \exists\,(\![[t,\otimes]]\!) \,;\, \sqcap$. *Suppose that for all* x *and* v,

$$\sqcap\{\, x \otimes n \mid n \in v \,\} \;=\; x \otimes (\sqcap v) \;.$$

Then

$$k \;=\; t \mathbin{\hat{\sqcap}} (\mathit{Inv}{+\!\!+} \,;\, \exists\,((L \times k) \,;\, \otimes) \,;\, \sqcap) \;.$$

It is readily verified that this instantiated theorem is applicable to the problem at hand: $\otimes$ is defined

$$x \otimes n \;=\; f\,x + n \;,$$

and $(+)$ distributes through the minimum function $(\sqcap)$. The resulting recursion equation for k underlies the well–known algorithm discussed by Knuth and Plass in [60]. To understand the workings of that algorithm, it may be helpful to translate the conclusion of the theorem into ordinary set theory. This is just a matter of expanding the definitions, and the outcome is displayed below:

$$k\,x \;=\; (t\,x) \sqcap (\sqcap\{\, a \otimes (k\,y) \mid a +\!\!+\, y = x \,\})\ .$$

Why is the instantiated theorem preferable to theorem 9.1.6? First of all, for its comprehensibility. The formulation given above can be understood by anyone acquainted with functional programming. This is not the case for theorem 9.1.6, which requires an understanding of F–algebras and cross–operators.

A second advantage of the instantiated theorem is that it is very close to a computer program. An efficient implementation of the recursion equation would tabulate all intermediate values during the computation, so that k is never evaluated twice on the same argument. Some functional programming languages include this tabulation as a language feature, called *memo–functions* [54, 70]. Translation of the above theorem into such a language is straightforward, and the result is a reasonably efficient program. Alternatively, one could code the tabulation by hand [11, 31] to obtain a truly efficient implementation in a more conventional programming language.

9.2.3 We now proceed to instantiate theorem 9.1.6 step by step. Throughout this proof, indices of natural transformations will be omitted, and we shall write 1 for the identity function. The first step is to instantiate the conclusion, showing that

$$\begin{array}{l} \mathit{Inv}[1, +\!\!+]\ ;\ \exists\,(F\,k\,;[t,\otimes])\ ;\ \sqcap \\ = \\ t\ \hat{\sqcap}\ (\mathit{Inv}{+\!\!+}\ ;\ \exists\,((1 \times k)\,;\oslash)\ ;\ \sqcap)\ . \end{array} \tag{9.1}$$

In proving this fact, we shall need the following law on the inverse image of coproducts:

$$\mathit{Inv}[g,\oplus] \;=\; (\mathit{Inv}\,g\,;\exists\,\iota_1)\ \hat{\cup}\ (\mathit{Inv}\oplus\,;\exists\,\iota_2) \tag{9.2}$$

where ι_1 and ι_2 are the coproduct injections and $\cup$ is set–union. We calculate from the left–hand side of equation 9.1:

$$\begin{array}{ll} & \mathit{Inv}[1, +\!\!+]\ ;\ \exists\,(F\,k\,;[t,\otimes])\ ;\ \sqcap \\ = & \{\ \text{eq. } (9.2)\ \} \\ & ((\mathit{Inv}1\ ;\exists\,\iota_1)\ \hat{\cup}\ (\mathit{Inv}{+\!\!+}\,;\exists\,\iota_2))\ ;\ \exists\,(F\,k\,;[t,\oslash])\ ;\ \sqcap \\ = & \{\ \cup : \exists \times \exists \rightarrow \exists,\ \exists\ \text{functor},\ \sqcap\ \text{selector}\ \} \\ & (\mathit{Inv}1\ ;\ \exists\,(\iota_1\,;F\,k\,;[t,\otimes])\ ;\ \sqcap)\ \hat{\sqcap}\ (\mathit{Inv}{+\!\!+}\ ;\ \exists\,(\iota_2\,;F\,k\,;[t,\otimes])\ ;\ \sqcap) \end{array}$$

$$
\begin{array}{ll}
= & \{ \text{ def. } F\text{, coproduct } \} \\
& (\mathit{Inv}1 \ ; \exists\, t \ ; \sqcap) \mathbin{\hat{\cap}} (\mathit{Inv}{+\!\!\!+} \ ; \exists\, ((1 \times k) ; \otimes) \ ; \sqcap) \\
= & \{ \ \mathit{Inv}1 = \{\text{-}\} \ \} \\
& (\{\text{-}\} \ ; \exists\, t \ ; \sqcap) \mathbin{\hat{\cap}} (\mathit{Inv}{+\!\!\!+} \ ; \exists\, ((1 \times k) ; \otimes) \ ; \sqcap) \\
= & \{ \text{ claim: see below } \} \\
& t \mathbin{\hat{\cap}} (\mathit{Inv}{+\!\!\!+} \ ; \exists\, ((1 \times k) ; \otimes) \ ; \sqcap)
\end{array}
$$

In the last step, it was claimed that

$$\{\text{-}\} ; \exists\, t ; \sqcap \ = \ t \, . \tag{9.3}$$

Here is the proof:

$$
\begin{array}{ll}
& \{\text{-}\} ; \exists\, t ; \sqcap \\
= & \{ \ \{\text{-}\} : 1 \rightarrow \exists \ \} \\
& t ; \{\text{-}\} ; \sqcap \\
= & \{ \ \sqcap \text{ selector } \} \\
& t
\end{array}
$$

We have now completed the instantiation of tbe conclusion of theorem 9.1.6, and it has been shown that

$$
\begin{array}{ll}
& \mathit{Inv}[1, +\!\!\!+] ; \exists\, (F k ; [t, \otimes]) ; \sqcap \\
= & \\
& t \mathbin{\hat{\cap}} (\mathit{Inv}{+\!\!\!+} \ ; \exists\, ((1 \times k) ; \otimes) \ ; \sqcap) \, ,
\end{array}
$$

as required. Our next task is to instantiate the applicability condition of theorem 9.1.6, which is the requirement that $[t, \otimes]$ distributes over $\sqcap$. More precisely, the goal is to try and simplify both sides of the equation

$$F\dagger ; \exists\, [t, \otimes] ; \sqcap \ = \ F\sqcap ; [t, \otimes] \, . \tag{9.4}$$

Mechanical application of the laws from paragraphs 6.2.5 to 6.2.10 shows that

$$F\dagger \ = \ (\{\text{-}\} + ((\{\text{-}\} \times 1) ; (\times)\dagger)) ; [\exists\, \iota_1, \exists\, \iota_2] \, .$$

For completeness, we shall carry out this calculation, despite its mechanical nature. Let K_L be the constant functor returning L. Then the functor F may be given in variable–free form as follows:

$$F \ = \ + \circ (K_L, \times \circ (K_L, 1)) \, . \tag{9.5}$$

Hence, bearing in mind that we decided to omit the object–indices of natural transformations, one may start pushing $(\text{-})\dagger$ into this expression for F:

$$\begin{array}{cl}
& F\dagger \\
= & \quad \{ \text{ eq. (9.5) } \} \\
& (+ \circ (K_L,\ \times \circ (K_L, 1)))\dagger \\
= & \quad \{ \ \dagger \text{ of composition (lemma 6.2.9) } \} \\
& (+(K_L,\ \times \circ (K_L, 1))\dagger)\ ;\ (+)\dagger \\
= & \quad \{ \ \dagger \text{ of } (+) \text{ (eq. 6.8) } \} \\
& (+(K_L,\ \times \circ (K_L, 1))\dagger)\ ;\ [\exists\,\iota_1, \exists\,\iota_2] \\
= & \quad \{ \ \dagger \text{ of product } (\text{-},\text{-}) \text{ (lemma 6.2.10) } \} \\
& (K_L\dagger\ +\ (\times \circ (K_L, 1))\dagger)\ ;\ [\exists\,\iota_1, \exists\,\iota_2] \\
= & \quad \{ \ \dagger \text{ of composition (lemma 6.2.9) } \} \\
& (K_L\dagger\ +\ (\times (K_L, 1)\dagger\,;(\times)\dagger))\ ;\ [\exists\,\iota_1, \exists\,\iota_2] \\
= & \quad \{ \ \dagger \text{ of product } (\text{-},\text{-}) \text{ (lemma 6.2.10) } \} \\
& (K_L\dagger\ +\ ((K_L\dagger \times 1\dagger)\,;(\times)\dagger))\ ;\ [\exists\,\iota_1, \exists\,\iota_2] \\
= & \quad \{ \ \dagger \text{ of } K_L \text{ (eq. 6.7) and 1 (eq. 6.7) } \} \\
& (\{\text{-}\} + ((\{\text{-}\} \times 1)\,;(\times)\dagger))\ ;\ [\exists\,\iota_1, \exists\,\iota_2]
\end{array}$$

So much for the unfolding of the cross–operator on F:

$$F\dagger \ = \ (\{\text{-}\} + ((\{\text{-}\} \times 1)\,;(\times)\dagger))\,;[\exists\,\iota_1, \exists\,\iota_2]\ . \qquad (9.6)$$

Using this fact, we can elaborate the left–hand side of the distributivity condition (9.4): $(F\dagger\ ;\ \exists[t, \otimes]\ ;\ \sqcap)$. However, it will be expedient to do a little subsidiary calculation first:

$$\begin{array}{cl}
& [\exists\,\iota_1, \exists\,\iota_2]\,;\exists\,[t,\otimes]\,;\sqcap \\
= & \quad \{ \text{ coproduct } \} \\
& [\exists\,\iota_1\,;\exists\,[t,\otimes]\,;\sqcap,\ \exists\,\iota_2\,;\exists\,[t,\otimes]\,;\sqcap] \\
= & \quad \{ \ \exists \text{ functor } \} \\
& [\exists\,(\iota_1\,;[t,\otimes])\,;\sqcap,\ \exists\,(\iota_2\,;[t,\otimes])\,;\sqcap] \\
= & \quad \{ \text{ coproduct } \} \\
& [\exists\,t\,;\sqcap,\ \exists\otimes\,;\sqcap]
\end{array}$$

Now it is easy to rewrite $(F\dagger\,;\exists[t,\otimes]\,;\sqcap)$ to a coproduct itself:

$$\begin{array}{cl}
 & F\dagger\,;\exists[t,\otimes]\,;\sqcap \\
= & \quad\{\text{ eq. (9.6) }\} \\
 & (\{\text{-}\}+((\{\text{-}\}\times 1)\,;(\times)\dagger))\,;[\exists\,\iota_1,\exists\,\iota_2]\,;\exists[t,\otimes]\,;\sqcap \\
= & \quad\{\text{ subsidiary calculation above }\} \\
 & (\{\text{-}\}+((\{\text{-}\}\times 1)\,;(\times)\dagger))\,;[\exists\,t\,;\sqcap,\ \exists\otimes\,;\sqcap] \\
= & \quad\{\text{ coproduct }\} \\
 & [\{\text{-}\}\,;\exists\,t\,;\sqcap,\ (\{\text{-}\}\times 1)\,;(\times)\dagger\,;\exists\otimes\,;\sqcap] \\
= & \quad\{\text{ eq. (9.3) }\} \\
 & [t,\ (\{\text{-}\}\times 1)\,;(\times)\dagger\,;\exists\otimes\,;\sqcap]
\end{array}$$

In short, it has been shown that

$$F\dagger\,;\exists\,[t,\otimes]\,;\sqcap \;=\; [t,\ (\{\text{-}\}\times 1)\,;(\times)\dagger\,;\exists\otimes\,;\sqcap]\ .$$

Recall that it is our goal to simplify both sides of the equation

$$F\dagger\,;\exists\,[t,\otimes]\,;\sqcap \;=\; F\sqcap\,;[t,\otimes]\ .$$

We have just rewritten the left–hand side to a coproduct; it stands to reason that we try and do the same with the right–hand side:

$$\begin{array}{cl}
 & F\sqcap\,;[t,\otimes] \\
= & \quad\{\text{ def. of } F\ \} \\
 & (1+(1\times\sqcap))\,;[t,\otimes] \\
= & \quad\{\text{ coproduct }\} \\
 & [t,(1\times\sqcap)\,;\otimes]
\end{array}$$

Summarising, we have sbown that $[t,\otimes]$ distributes through $\sqcap$ if and only if

$$[t,\ (\{\text{-}\}\times 1)\,;(\times)\dagger\,;\exists\otimes\,;\sqcap] \;=\; [t,\ (1\times\sqcap)\,;\otimes]\ .$$

By the universal property of the coproduct, this equation can be further simplified to get

$$(\{\text{-}\}\times 1)\,;(\times)\dagger\,;\exists\otimes\,;\sqcap \;=\; (1\times\sqcap)\,;\otimes\ .$$

We could leave it at this, but again it may be helpful to translate the expressions into ordinary set–theoretic terms. Here is the result:

$$\sqcap \{ x \otimes n \mid n \in v \} \;=\; x \otimes (\sqcap v) \;,$$

for all x and v. Indeed, this was the condition originally stated in the instantiation of theorem 9.1.6.

Some readers may find the preceding proof ludicrously long and laborious. I sympathise with this opinion, and in fact it is possible to do this type of instantiation completely mechanically. In fact, I have written a computer program in OBJ3 [45] that performs this task for a restricted class of examples.

9.3 A more Practical Theorem

One could distinguish between three different stages in the application of dynamic programming. First, the problem at hand is phrased as an instance of the generic specification. Sometimes this is easy (as in the text formatting example) but it may also be quite difficult. The second stage is to verify that the applicability conditions are met: one has to verify that M is a selector, and that h distributes over M. Finally, one migbt wish to instantiate the abstract result because it has to be implemented in a conventional programming language. In the preceding section, it was shown that such instantiations can be performed in an entirely mechanical fashion.

To ease the application of dynamic programming is therefore to simplify the first two stages of the application process: specification and verification. These two stages are deeply intertwined. For example, in the text-formatting problem, we introduced the fictitious value *infinity* in the specification to make sure that $\sqcap$ is a selector. In this section, it will be shown how relation totalisers can be used to ease both specification and verification of dynamic programming.

9.3.1 From now on, it is assumed that $\mathcal{E}$ is a Boolean topos with a natural numbers object. The existence of a natural numbers object implies that all polynomial functors have an initial algebra. Therefore, we assume that the endofunctor $F : \mathcal{E} \to \mathcal{E}$ is polynomial. As before, $S : FB \to B$ is a relational F^*–algebra, and $h : FA \to A$ is an F–algebra. Finally, R is assumed to be a relation $A \to A$. The new generic specification reads as follows:

$$K \;=\; (\mathit{Inv}(\!|S|\!) \,;\, \exists\,(\!|h|\!) \,;\, \mathit{max}\, R)^{\smile} \,.$$

In the next two lemmas, it will be shown that this is an instance of our earlier generic specification.

9.3.2 Lemma

$$\llparenthesis h \rrparenthesis \,;\, \eta_A \;=\; \llparenthesis \overline{F}_A \,;\, Eh \rrparenthesis$$

Proof By proposition 6.3.5, it suffices to show that

$$h \,;\, \eta_A \;=\; F\eta_A \,;\, \overline{F}_A \,;\, Eh \;.$$

Here is a proof of that identity:

$$
\begin{array}{ll}
 & F\eta_A \,;\, \overline{F}_A \,;\, Eh \\
= & \quad \{ \text{ property of } \overline{F} \text{ (par. 7.3.4) } \} \\
 & \eta_{FA} \,;\, Eh \\
= & \quad \{ \; \eta : \mathcal{E} \to E \; \} \\
 & h \,;\, \eta_A
\end{array}
$$

9.3.3 Lemma

$$(\mathit{Inv}\llbracket S \rrbracket \,;\, \exists\, \llparenthesis h \rrparenthesis \,;\, \mathit{max}\, R)^{\sim} \;=\; \mathit{Inv}\llbracket S \rrbracket \,;\, \exists\, \llparenthesis \overline{F}_A \,;\, Eh \rrparenthesis \,;\, \mathit{sel}\, R$$

Proof

$$
\begin{array}{ll}
 & (\mathit{Inv}\llbracket S \rrbracket \,;\, \exists\, \llparenthesis h \rrparenthesis \,;\, \mathit{max}\, R)^{\sim} \\
= & \quad \{ \; (\mathit{Inv}\llbracket S \rrbracket \,;\, \exists\, \llparenthesis h \rrparenthesis) \text{ map, relation totaliser (prop. 7.3.2) } \} \\
 & \mathit{Inv}\llbracket S \rrbracket \,;\, \exists\, \llparenthesis h \rrparenthesis \,;\, \widetilde{\mathit{max}}\, R \\
= & \quad \{ \; \eta_A \text{ monic } \} \\
 & \mathit{Inv}\llbracket S \rrbracket \,;\, \exists(\llparenthesis h \rrparenthesis \,;\, \eta_A \,;\, \eta_A^{\circ}) \,;\, \widetilde{\mathit{max}}\, R \\
= & \quad \{ \; \exists \text{ functor } \} \\
 & \mathit{Inv}\llbracket S \rrbracket \,;\, \exists(\llparenthesis h \rrparenthesis \,;\, \eta_A) \,;\, \exists\, \eta_A^{\circ} \,;\, \widetilde{\mathit{max}}\, R \\
= & \quad \{ \text{ def. selector of } R \; \} \\
 & \mathit{Inv}\llbracket S \rrbracket \,;\, \exists(\llparenthesis h \rrparenthesis \,;\, \eta_A) \,;\, \mathit{sel}\, R \\
= & \quad \{ \text{ lemma 9.3.2 } \} \\
 & \mathit{Inv}\llbracket S \rrbracket \,;\, \exists\, \llparenthesis \overline{F}_A \,;\, Eh \rrparenthesis \,;\, \mathit{sel}\, R
\end{array}
$$

9.3.4 Theorem *Consider the specification*

$$K \;=\; (\mathit{Inv}⦇S⦈ \,;\, \exists\, ⦇h⦈ \,;\, \mathit{max}\, R)^{\smile} \,.$$

Suppose that

- *the relation R is a well–founded preorder, and*
- *the map h is monotonic with respect to R,*

$$\begin{tikzcd}
A \arrow[d, "R"'] & FA \arrow[l, "h"'] \arrow[d, "F^{\star}R"] \\
A \arrow[r, "h^{\circ}"'] \arrow[ur, phantom, "\boxed{?}"] & FA
\end{tikzcd}$$

where $\boxed{?}$ *is* $(\subseteq)$, $(\supseteq)$ or $(=)$.

Then K satisfies

$$\begin{tikzcd}
B \arrow[r, "\mathit{InvS}"] \arrow[d, "K"'] & PFB \arrow[d, "\exists V"] \\
EA \arrow[ur, phantom, "\boxed{?}"] & PEA \arrow[l, "\mathit{sel}R"]
\end{tikzcd}$$

where V is the composite

$$FB \xrightarrow{F^{\star}K} FEA \xrightarrow{\overline{F}_A} EFA \xrightarrow{Eh} EA$$

Proof Immediate from lemma 9.3.3, theorem 8.2.3, theorem 8.3.2, and theorem 9.1.6.

9.3.5 As an application, let us return to the text formatting example of the preceding section. To avoid the introduction of a list of infinite length, we only considered the problem of determining the minimum amount of white space, and we did not consider how such an optimal layout itself might be constructed. The new theorem makes it easy to reason about a fictitious list of infinite length, so now we can address the problem of constructing an optimal layout. Here is the specification:

$$K \;=\; (\mathit{Inv}(\llparenthesis[L, +\!\!+]\rrparenthesis) \,;\, \mathit{max}\, R)^{\smile} \,,$$

where R is the preorder

$$R \;=\; \llparenthesis[t, \otimes]\rrparenthesis ; \geq ; \llparenthesis[t, \otimes]\rrparenthesis^{\circ} \,.$$

Note that this preorder is well–founded by proposition 8.1.6. Write τ for the initial F–algebra. We have

$$K \;=\; (\mathit{Inv}(\llparenthesis[L, +\!\!+]\rrparenthesis) \,;\, \exists\, \llparenthesis\tau\rrparenthesis \,;\, \mathit{max}\, R)^{\smile} \,,$$

because $\llparenthesis\tau\rrparenthesis$ is the identity morphism T. To apply the new result about dynamic programming, we should verify that

$$\begin{tikzcd}
T \arrow[d, "R"'] & (L + (L \times T)) \arrow[l, "\tau"'] \arrow[d, "F^{\star}R"] \\
T \arrow[r, "\tau^{\circ}"'] \arrow[ur, phantom, "\supset"] & L + (L \times T)
\end{tikzcd}$$

semi–commutes. By proposition 8.3.3, it suffices to show that

$$\begin{tikzcd}
\mathrm{N} \arrow[d, "(\geq)"'] & (L + (L \times \mathrm{N})) \arrow[l, "{[t, \otimes]}"'] \arrow[d, "F^{\star}(\geq)"] \\
\mathrm{N} \arrow[r, "{[t, \otimes]^{\circ}}"'] \arrow[ur, phantom, "\supset"] & L + (L \times \mathrm{N})
\end{tikzcd}$$

semi–commutes. This last inequation may be proved as follows:

$$\begin{aligned}
& F^{\star}(\geq) \subset [t, \otimes] \,;\, (\geq) \,;\, [t, \otimes]^{\circ} \\
\Leftrightarrow \quad & \{\ [t, \otimes] \text{ map }\ \} \\
& F^{\star}(\geq) \,;\, [t, \otimes] \subset [t, \otimes] \,;\, (\geq) \\
\Leftrightarrow \quad & \{\ \text{def. } F\text{, coproduct in } \mathit{Rel}(\mathit{Set})\ \} \\
& t \subset t \,;\, (\geq) \quad \text{and} \quad (L \times^{\star} (\geq)) \,;\, (\otimes) \subset (\otimes) \,;\, (\geq)
\end{aligned}$$

The first conjunct is immediate from the fact that $(\geq)$ is reflexive. It remains to show that $(\otimes) \,;(\geq) \supset (L \times^{\star} (\geq))\,;(\otimes)$. This is a simple consequence of the fact that addition is monotonic with respect to $(\geq)$:

$$\begin{array}{ll}
 & (\otimes)\,;(\geq) \\
= & \quad \{ \text{ def. } \otimes \ \} \\
 & (f \times \mathrm{N})\,;(+)\,;(\geq) \\
\supset & \quad \{ \ + \text{ monotonic } \} \\
 & (f \times \mathrm{N})\,;(\mathrm{N} \times^{\star} (\geq))\,;(+) \\
= & \quad \{ \ \times \text{ relator, def. } \otimes \ \} \\
 & (L \times^{\star} (\geq))\,;(\otimes)
\end{array}$$

The proof that addition is monotonic is omitted.

9.3.6 Although the theorem about dynamic programming is adequate for typical applications like text formatting, knapsack and string–to–string correction, there are problems that need a slightly different approach. An example of such a problem is bracketing a sequence of matrices for multiplication; we already discussed it in the first part of this thesis (section 2.3). The proof of the theorem that was stated there is very similar to the proof given here; the details are omitted.

10 Discussion

It is generally recognised that the calculus of relations is a valuable tool in developing programs from specifications. Until recently, most of the work on relations in computing science either used a set theoretic approach, or the axiomatisation of relation algebras proposed by Tarski [93]. The categorical viewpoint reconciles the axiomatic approach to relations with the set theoretic one, in that it shows how set theory (or rather topos theory) can be defined in terms of relations. In this thesis, we have tried to demonstrate the advantages of this viewpoint for applications in program construction. The main advantage is, I believe, the smooth generalisation of functional concepts like *fold* to relations.

Backhouse *et al.* have shown that such a generalisation is also possible in a more traditional form of the relational calculus [6]. There is however a subtle difference: Backhouse defines all categorical constructions at the level of relatious, and then shows that they satisfy the expected properties in the subcategory of maps. In my work, the process goes the other way round: constructions in the subcategory of maps are extended to the category of relations. Which approach you prefer depends on whether you see functions or relations as the more basic entity. But does it really matter whether one starts with maps or with relations? The answer is no, because regular categories and unitary tabular allegories are equivalent. This is the attraction of category theory: it uncovers the *structure* of the calculus of relations, not only its theorems.

In this thesis, we have made ample use of that structural aspect of the theory: typical instances are the definition of cross–operators and the generalisation of *fold* to relations. In the remainder of this discussion, we shall briefly present three open problems that are concerned with the structure of relations. This is not to say that further research on dynamic programming is unimportant; it is however much clearer in that area where the research is heading, and we already discussed some of those future directions in part I.

10.1 Which Functors preserve Division?

In the proof that monotonicity implies distributivity, we used the fact that polynomial functors preserve strict left–division. Actually, it was this property that motivated the restriction to polynomial functors. It would he much better to have a necessary condition which tells when a relator preserves strict left–division. There exists such a condition for the preservation of intersection: the relator should preserve pullbacks. That result is intuitively obvious, because intersection is defined in terms of pullbacks. To achieve a similar theorem for strict left–division, one should identify the elementary categorical construction that underlies its definition. The following proposition might be of help in achieving this goal.

Proposition *Let F be a relator. Then $F^\star$ preserves strict left–division iff $F^\star$ preserves right–quotients of entire relations.*

For any topos, there exists an adjunction between the topos itself and its category of entire relations. Motivated by the above result, I briefly tried to define right-quotients of entire relations in terms of this adjunction. This may be done, but it did not lead to any further insight.

10.2 Does every Topos have a Weak Relation Totaliser?

The definition of *relation totaliser* proposed in chapter 4 is unsatisfactory, because not every topos has one. While developing the results about relation totalisers, I considered an alternative definition which is less restrictive than the one adopted in this thesis. The disadvantage of this alternative definition is that it does not seem to give a nice calculus. Below we shall briefly summarise some results which indicate that it might still be worthwhile to pursue the topic further. To keep the account short, all proofs are omitted.

Consider the category of relations over a regular category. A *weak relation totaliser* is a collection of monic maps

$$\eta_A : A \rightarrowtail EA$$

which satisfies the following pseudo–universal property: for every relation $R : A \to B$ there exists an entire relation

$$\tilde{R} : A \to EB$$

such that

$$\begin{array}{ll} & S ; \eta_B{}^\circ \;=\; R \text{ and } {}^<R ; S \subset R ; \eta_B \\ \Leftrightarrow & \\ & S \subset \tilde{R} \end{array}$$

for all relations $S : A \to EB$. The question is now which toposes do have a weak relation totaliser. To discuss that qnestion, we need a definition of the *mild axiom of choice*.

Let $\mathcal{E}$ be a regular category. Say that $\mathcal{E}$ satisfies the *mild axiom of choice* if for every relation R, there exist a cover e and a simple relation F such that

$$F \ \subset \ e\,;R \ \text{ and } \ {}^{<}(e\,;R) \subset {}^{<}F \ .$$

Proposition *A topos $\mathcal{E}$ satisfies the mild axiom of choice if $E : \mathcal{E} \to \mathcal{E}$ preserves epics.*

For example, the topos $Set^{\rightarrow}$ of commuting squares does satisfy the mild axiom of choice. The usual form of the axiom of choice in category theory says that all epics have a left–inverse. Tbe topos of commuting squares does *not* satisfy the usual axiom of choice. To the best of my knowledge, it is unknown whether E preserves epics in every topos. The next theorem shows why it might be interesting to resolve that issue:

Theorem *Let $\mathcal{E}$ be a topos. Then the simple–relation classifier is a weak relation totaliser iff $\mathcal{E}$ satisfies the mild axiom of choice.*

10.3 What are Selectors?

Our discussion of selectors focussed on the pragmatic aspects of their use in dynamic programming, and I believe that from this point of view the relevant questions have been answered. However, the construction of selectors from preorders $sel(\text{-})$ only covers a very small class: there are many selectors that do not arise in this way, for example $\bigcup$ itself. This leads to the question what selectors really are: do they form a category? Is there an alternative description of this category that gives more insight in their precise nature? First, we need some more terminology. Let $\mathcal{E}$ be a topos.

For any relation $R : A \to A$ in $Rel(\mathcal{E})$, one can define the *supremum* relation $sup\, R : PA \to A$ by

$$sup\, R \ = \ R{\uparrow} \ ; \ min\, R \ .$$

A partial order is said to be *cocomplete* if its supremum relation is a map. The cocomplete partial orders in $\mathcal{E}$ form a category, where the objects are cocomplete partial orders and the arrows are sup–preserving morphisms. An arrow $f : A \to B$ is said to be a *sup–preserving* morphism from $R : A \to A$ to $S : B \to B$ if

$$sup\, R \ ; \ f \ = \ \exists f \ ; \ sup\, S \ .$$

The category of *functional selectors* is defined as follows. Its objects are selectors which are maps, and its arrows are selection-preserving morphisms in $\mathcal{E}$. Let $h : PA \to A$ and $k : PB \to B$ be selectors in $\mathcal{E}$. A *selection-preserving* morphism from h to k is an arrow $f : A \to B$ such that

$$h\,;f \;=\; \exists f\,;k\;.$$

Alternatively, the category of functional selectors could be defined as the Eilenberg-Moore category of the power monad.

Theorem (Mikkelsen [71], p. 36) *In any topos, the category of cocomplete partial orders is isomorphic to the category of functional selectors.*

To gain a proper understanding of what selectors really are, it would be helpful to have a similar theorem for arbitrary selectors. Unfortunately, time constraints did not allow me to investigate the issue in any detail.

Bibliography

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison–Wesley, 1974.

[2] A. Apostolico, M.J. Atallah, L.L. Larmore, and S. McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM Journal on Computing*, 19(5):968–988, 1981.

[3] M.A. Arbib and E.G. Manes. *Arrows, Structures and Functors: The Categorical Imperative*. Academic Press, 1975.

[4] R.C. Backhouse. Makiug formality work for us. *EATCS Bulletin*, 38:219–249, 1989.

[5] R.C. Backhouse, editor. *EURICS Workshop* on *Calculational Theories of Program Structure*, 23–27 September 1991.

[6] R.C. Backhouse, E. Voermans, and J.C.S.P. van der Woude. A relational theory of datatypes. Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. Appears in [5], 1991.

[7] M. Barr. Exact categories. In *Exact Categories and Categories of Sheaves*, volume 236 of *Lecture Notes in Mathematics*, pages 1–120. Springer–Verlag, 1970.

[8] J. Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer–Verlag, 1969.

[9] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[10] R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43(2–3):123–147, 1986.

[11] R.S. Bird. Tabulation techniques for recnrsive programs. *Computing Surveys*, 12(4):403–417, December 1980.

[12] R.S. Bird. Transformational programming and the paragraph problem. *Science of Computer Programming*, 6:159–189, 1986.

[13] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, volume 36 of *NATO ASI Series F*, pages 3–42. Springer–Verlag, 1987.

[14] R.S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*, volume 55 of *NATO ASI Series F*, pages 151–216. Springer–Verlag, 1989.

[15] R.S. Bird. A calculus of functions for program derivation. In D.A. Turner, editor, *Research Topics in Functional Programming*, University of Texas at Austin Year of Programming Series, pages 287–308. Addison–Wesley. 1990.

[16] R.S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, 12:93–104, 1989.

[17] R.S. Bird and O. de Moor. Nub theory. Draft, 1991.

[18] R.S. Bird and P. Wadler. *Introduction to Functional Programming*. Prentice–Hall, 1988.

[19] T. Brook. *Order and Recursion in Topoi*, volume 9 of *Notes on Pure Mathematics*. Australian National University, Canberra, 1977.

[20] F. Cajori. *A History of Mathematical Notations. Volume I : Notations in Elementary Mathematics*. The Open Court Publishing Company, 1928.

[21] F. Cajori. *A History of Mathematical Notations. Volume II : Notations Mainly in Higher Mathematics*. The Open Court Publishing Company, 1929.

[22] A. Carboni and S. Kasangian. Bicategories of spans and relations. *Journal of Pure and Applied Algebra*, 33:259–267, 1984.

[23] A. Carboni, G.M. Kelly, and R.J. Wood. A 2–categorical approach to geometric morphisms, I. Research Report 89–19, Department of Pure Mathematics, The University of Sydney, NSW 2006, Australia, 1989. ISSN 1033–2359.

[24] A. Carboni and G. Rosolini. The free regular category on a left exact one. In preparation, 1991.

[25] A. Carboni and R. Street. Order ideals in categories. *Pacific Journal of Mathematics*, 124(2):275–288, 1986.

[26] A. Carboni and R.F.C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49:11–32, 1987.

[27] R.J. Casimir. Program inversion. Technical Report AIV-80-10, Vakgroep AIV, Erasmus Universiteit, Postbus 1730, 3000 DR Rotterdam, The Netherlands, July 1980.

[28] W. Chen and J.T. Udding. Program inversion: More than fun! *Science of Computer Programming*, 15(1):1–13, 1990.

[29] R. Cockett. Personal communication. 1991.

[30] R. Cockett and T. Fukushima. Draft: About Charity. Dept. of Compnter Science, University of Calgary, Calgary, Alberta, Canada, 1991.

[31] N.H. Cohen. Characterization and elimination of redundancy in recursive programs. In *6th ACM Annual Symposium on Principles of Programming Languages*, pages 143–157. Association for Computing Machinery, 1979.

[32] E.V. Denardo. *Dynamic Programming — Models and Applications*. Prentice–Hall, 1982.

[33] E.W. Dijkstra. Program inversion. In F.L. Bauer and M. Broy, editors, *Program Construction*, volume 69 of *Lecture Notes in Computer Science*, pages 54–57. Springer–Verlag, 1979.

[34] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer–Verlag, 1990.

[35] J.G. Ecker and M. Kupferschmid. *Introduction to Operations Research.* John Wiley, 1988.

[36] S. Eilenberg and J.B. Wright. Automata in general algebras. *Information and Control*, 11(4):452–470, 1967.

[37] A.J. Field and P.G. Harrison. *Functional Programming.* International computer science series. Addison–Wesley, 1988.

[38] M.M. Fokkinga. An exercise in transformational programming: Backtracking and branch–and–bound. *Science of Computer Programming*, 16:19–48, 1991.

[39] P.J. Freyd. Personal communication. 1991.

[40] P.J. Freyd and A. Ščedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North–Holland, 1990.

[41] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theoretical Computer Science*, 64:107–118, 1989.

[42] P. Gardiner. Personal communication. 1990.

[43] J.A. Goguen. Personal communication. 1989.

[44] J.A. Goguen and J. Meseguer. Correctness of recursive parallel nondeterministic flow programs. *Journal of Computer and System Sciences*, 27(2):268–290, 1983.

[45] J.A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, Computing Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, USA, Augnst 1988.

[46] R. Goldblatt. *Topoi — The Categorial Analysis of Logic*, volume 98 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1986.

[47] P.A. Grillet. Regular categories. In *Exact Categories and Categories of Sheaves*, volume 236 of *Lecture Notes in Mathematics*, pages 121–222. Spriuger-Verlag, 1970.

[48] P.G. Harrison and H. Khoshnevisan. On the synthesis of function inverses. Research Report DOC 90/4, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, England, February 1990.

[49] P. Helman. The principle of optimality in the design of efficient algorithms. *Journal of Mathematical Analysis and Applications*, 119:97–127, 1986.

[50] P. Helman. A common schema for dynamic programming and branch-and-bound algorithms. *Journal of the ACM*, 36(1):97–128, January 1989.

[51] P. Helman and A. Rosenthal. A comprehensive model of dynamic programming. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):319–334, 1985.

[52] D.S. Hirschberg and L.L. Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987.

[53] P.J. Huber. Homotopy theory in general categories. *Math. Annalen*, 144:361–385, 1961.

[54] J. Hughes. Lazy memo-functions. In J.P. Jouannaud, editor, *Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 130–146. Springer-Verlag, 1985.

[55] J. Jeuring. Algorithms from theorems. In M. Broy and C.B. Jones, editors, *Programming Concepts and Methods*, pages 247–266. North–Holland, 1990.

[56] P.T. Johnstone. *Topos Theory*. Academic Press, 1977.

[57] G. Jones. Designing circuits hy calculation. Technical Report PRG–TR–10–90, Programming Research Group, 11 Kehle Road, Oxford OX1 3QD, England, 1990.

[58] R.M. Karp and M. Held. Finite–state processes and dynamic programming. *SIAM Journal on Applied Mathematics*, 15(3):693–718, 1967.

[59] H. Kleisli. Every standard construction is induced by a pair of adjoint functors. *Proceedings of the American Mathematical Society*, 16:544–546, 1965.

[60] D.E. Knuth and M.F. Plass. Breaking paragraphs into lines. *Software: Practice and Experience*, 11:1119–1184, 1981.

[61] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[62] D.J. Lehmann and M.B. Smyth. Algebraic specification of data types: A synthetic approach. *Mathematical Systems Theory*, 14:97–139, 1981.

[63] B. Louka and M. Tchuente. Dynamic programming on two–dimensional systolic arrays. *Information Processing Letters*, 29:97–104, 1988.

[64] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14:255–279, 1990.

[65] E.G. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer–Verlag, 1975.

[66] E.G. Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monographs in Computer Science. Springer–Verlag, 1986.

[67] C.E. Martin. Preordered categories and predicate transformers. D.Phil. thesis. Programming Research Group, Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, October 1991.

[68] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker, M. Hazewinkel, and J.K. Lenstra, editors, *Mathematics and Computer Science*, volume 1 of *CWI Monographs*, pages 3–42. North–Holland, 1987.

[69] K. Mehlhorn. *Data Structures and Algorithms*. EATCS Monographs on Theoretical Computer Science. Springer–Verlag, 1984. (3 volumes).

[70] D. Michie. "Memo" functions and machine learning. *Nature*, 218:19–22, April 1968.

[71] C.J. Mikkelsen. Lattice theoretic and logical aspects of elementary topoi. Various Publications Series 25, Matematisk Institut, Aarhus Universitet, NY Munkegade, DK-8000 Aarhus C, Denmark, 1976.

[72] O. de Moor. Context-free language recoguition. International Summer School on Constructive Algorithmics, Hollum, Ameland, The Netherlands, 1989.

[73] O. de Moor. Categories, relations and dynamic programming. Technical Report PRG–TR–18–90, Programming Research Gronp, 11 Keble Road, Oxford OX1 3QD, England, 1990.

[74] O. de Moor and R.S. Bird. List partitions, to appear in Formal Aspects of Computing. Programming Research Group, 11 Keble Road, Oxford OX1 3QD, England, 1989.

[75] C.C. Morgau. *Programming from Specifications*. Prentice–Hall, 1990.

[76] J.F. Myoupo. Dynamic programming on linear pipelines. *Information Processing Letters*, 39:333–341, 1991.

[77] G.D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[78] A. Poigné. A note on distributive laws and power domains. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Programming*, volume 240 of *Lectnre Notes in Computer Science*, pages 252–265. Springer–Verlag, 1986.

[79] W. Rytter. On efficient parallel computations for some dynamic programming problems. *Theoretical Computer Science*, 59:297–307, 1988.

[80] G. Schmidt. Programs as partial graphs I: Flow equivalence and correctness. *Theoretical Computer Science*, 15:1–25, 1981.

[81] H. Schubert. *Categories*. Springer–Verlag, 1970.

[82] R. Sedgewick. *Algorithms*. Addison–Wesley, 1983.

[83] M. Sheeran. Describing hardware algorithms in Ruby. In David *et al.*, editors, *IFIP WG 10.1 workshop on Concepts and Characteristics of Declarative Systems, Budapest 1988*. North–Holland, 1989.

[84] M. Sheeran. Categories for the working hardware designer. In M. Leeser and G. Brown, editors, *Workshop on Hardware Specification, Verification and Synthesis: Mathematical Aspects. Cornell University 1989*, volume 408 of *Lecture Notes in Computer Science*, pages 380–402. Springer–Verlag, 1990.

[85] D.R. Smith. Top–down synthesis of divide–and–conquer algorithms. *Artificial Intelligence*, 27:43–96, 1985.

[86] D.R. Smith. Applications of a strategy for designing divide–and–conquer algorithms. *Science of Computer Programming*, 18:213–229, 1987.

[87] D.R. Smith. Structure and design of global search algorithms. Technical Report KES.U.87.12, Kestrel Institute, 1801 Page Mill Road, Palo Alto, CA 94304, July 1988. To appear in Acta Informatica.

[88] D.R. Smith. Kids: A semiautomatic program development system. *IEEE Transactions on Software Engineering*, 16(9):1024–1043, 1990.

[89] D.R. Smith. Structure and design of problem reduction generators. In B. Möller, editor, *Proc. of the IFIP TC2 Working Conference on Constructing Programs from Specifications*. North–Holland, 1991.

[90] D.R. Smith and M.R. Lowry. Algorithm theories and design tactics. *Science of Computer Programming*, 14:305–321, 1990.

[91] M. Sniedovich. A new look at Bellman's principle of optimality. *Journal of Optimization Theory and Applications*, 49(1):161–176, April 1986.

[92] J.M. Spivey. A functional theory of exceptions. *Science of Computer Programming*, 14:25–42, 1990.

[93] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[94] D.A. Turner. Miranda — a non–strict functional language with polymorphic types. In P. Henderson and D.A. Turner, editors, *Proc. Conference on Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer–Verlag, 1985.

[95] A.J.M. van Gasteren. *On the shape of mathematical arguments*, volume 445 of *Lecture Notes in Computer Science*. Springer–Verlag, 1990.

[96] P. Wadler. Comprehending monads. In G. Kahn, editor, *ACM Conference on Lisp and Functional Programming*, pages 61–78. ACM Press, 1990.

[97] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168-173, 1974.

Index

adjoint arrow, 54
adjunction, 15, 69, 110
Aho, 32
algebra, 87
- initial, 5, 87
- relational, 89

algorithm theory, 34
allegory, 64
- power, 93

anti–symmetric, 123
Arbib, 49
associative law, 70
asymmetric equality, 44

Back, 37
Backhouse, 66, 75, 113, 161
Barr, 51
Beck, 76
Bellman, 3, 32
binary trees, 26
Bird, i, v, 38
Boolean, 15, 113
bottom–up evaluation, 32
bracketing, 26
Brook, 38, 144

Cajori, 43
Carboni, v, 43
cargo, 24
cartesian, 47
categorical programming, 36
category of relations, 52
Charity, 36
choice function, 33
classifier, 15
- simple relation, 108

cocartesian, 73
Cockett, 36
cocomplete, 73
coequaliser, 51
commuting squares, 111
comparison functor, 72
complete, 47
concatenation, 4
concave, 36
conjugate kernel, 94
containment, 101
- naturality, 101

converse, 5, 53
coproduct, 73
coreflective, 107
coreflexive, 65
cover, 49, 53
cover–monic factorisation, 49
covered by, 50
cross–operator, 9, 75
- composition, 85
- example, 82
- naturality, 85
- product, 86

Denardo, 32
diagonal fill–in, 49
direct image, 49
discrete decision process, 33
discrete optimisation problem, 33
distribute, 16, 137
distributive law, 9, 76
divide–and–conquer, 3, 35
division, 14
- left, 93
- right, 91
- strict, 94

domain, 14, 65
dynamic programming, 32

basic theorem, 11, 148
divide-and-conquer, 149
instantiation, 156
practical theorem, 18, 158

edit distance, 21
Egli–Milner order, 105
Eilenberg, 10, 89
element, 90
entire, 14, 53, 64
existential image, 106
epsiloff, 69
equaliser, 49
everywhere defined, 14, 53
existential image, 6, 9, 69
relator, 98
extend, 8, 62, 76
extremal epic, 49

factorisation, 49
factors, 47
faithful, 53
fictitious, 13
fold, 5, 88
format, 3
Freyd, v, 43
frontier, 27
Fukushima, 36
function, 54
functor
extended, 56

Gardiner, v, 63
Gasteren, 117
generic specification, 7, 145
Goguen, v, 87, 146
Goldblatt, 112
graph, 53
greatest lower bound
subobject, 47
greedy, 38
Grillet, 51

hardware, 37
has-element, 90
Held, 33
Helman, 33
Hoare, v, 105
homomorphism, 87
relational, 90
Hopcroft, 32
Huber, 71

identity, 44
image, 49
existential, 69
imperative, 36
infinity, 13
initial, 87
initial algebra, 17
existence, 91
relational, 10
internal intersection, 93
inverse, 53
inverse image, 5, 145
involution, 53

Jones, 37
Joyal, 51

Karp, 33
Kelly, 43
Kleish, 71
category, 71
knapsack, 24

Lambek, 89
Lawvere, 73, 108
layout, 3, 149
least, 14
least upper bound
of relations, 74
list, 5, 87
lists, 22
load, 24
Lowry, 34

Luk, 37

Malcolm, 88
Manes, 49, 76
map, 6, 54, 64
Martin, v, 37
matrix multiplication, 26
maximum
  existential image, 124
  of set, 122
  simple, 123
  union, 126
mechanisation, 34
Meertens, i, v
memoisation, 7, 152
Meseguer, 87
mild axiom of choice, 162
minimum, 14
modular law, 64
monad, 70
  defined by adjunction, 71
  simple relation, 118
monic, 47, 54
monotonic, 16, 137
Morgan, 37

natural numbers object, 17, 88, 91
  Boolean, 144
  well-bounded, 144
nearly preserve, 8
non-associative regular expression, 33
notation, 43

OBJ3, 35, 156
object problem, 19
operations research, 32
optimal, 3
order-enriched, 52

P-RAM, 37
paragraph, 3, 149
parallel, 37
parenthesis, 26
partial arrow, 30, 54
partial arrow classifier, 15
partial order, 123
Poigné, 76
polynomial, 16, 90
  strict division, 95
power
  adjunction, 69
  allegory, 93
  monad, 71
  object, 69
  transpose, 5
    of functor, 84
power allegory
  topos, 93
power transpose, 72
predicate trausformer, 37
preorder, 15, 123
preserve
  covers, 57
  domain, 66
  intersection, 60
  nearly, 8, 58
  pullbacks, 8, 60
    up to image, 58
  regular epics, 8
  strict division, 95, 161
  tabulations, 60, 91
principle of optimality, 3, 16
problem reduction generator, 35
problem theory, 34

quotient, 14
  left, 93
  right, 91

reciprocal, 53
recursion, 7, 35
reduce, 5, 88
refinement calculus, 37
reflexive, 123
regular

category, 51
epic, 51
regular category, 64
regular epic, 8
relation, 52
totaliser, 15, 111
weak, 162
relator, 8, 59, 98
example, 59
polynomial, 16
reverse, 53
Rosenthal, 33

Scedrov, 43
Schroeder, 113
Schubert, 72
selector, 13, 128
functional, 164
of preorder, 129
semi-lattice, 48
sequence, 5, 87
sequences, 22
sequential decision process, 33
set-valued functions, 72
Sheeran, 37
shunt, 113
simple, 54, 64, 73
singleton, 69
smash, 119
Smith, v, 34
Smyth, 105
Spivey, 73
splitting, 36
string-to-string, 21
subfunctor, 110
subobject, 47
extended, 51
functor, 48
subproblem, 3
subsequence, 22
Swierstra, v
systolic, 37

tabular, 66
tabulation, 7, 32, 51, 56, 152
terminator, 88
text formatting, 3, 149
Tierney, 73, 108
topos, 69
Boolean, 15, 113, 144
relation totaliser, 114
power allegory, 93
total, 14, 53
totaliser, 15, 111
transform, 21
transitive, 123
transpose, 53
triangular identity, 69
typed factor, 94

Ullman, 32
union
of relations, 74
of sets, 69
unit, 66
unit law, 70
upper bound
of set, 122

value problem, 19
vessel, 24
view, 35

Wadler, 70
weakest postspecification, 94
well-bounded, 123, 125
well-founded, 15
Wells, v
Wood, 43
Wrigbt, 10, 89