Universidade do Minho
Escola de Engenharia
Departamento de Informática

Alexandre Mendonça Pinho

Greedy and Dynamic Programming by Calculation

May 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Alexandre Mendonça Pinho

Greedy and Dynamic Programming by Calculation

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
José Nuno Oliveira

May 2022

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Professor José Nuno Oliveira, for the guidance and encouragement he has provided through each stage of this project. His expertise and continuous support were invaluable contributions.

I would also like to thank my friends, colleagues and professors at University of Minho for all the support and shared experiences in these past years.

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

The mathematical study of the greedy algorithm provides a blueprint for the study of Dynamic Programming (DP), whose body of knowledge is largely unorganized, remaining obscure to a large part of the software engineering community. This study aims to structure this body of knowledge, narrowing the gap between a purely example-based approach to DP and its scientific foundations. To that effect, matroid theory is leveraged through a *pointfree* relation algebra, which is applied to greedy and DP problems. A catalogue of such problems is compiled, and a broad characterization of DP algorithms is given. Alongside, the theory underlying the thinning relational operator is explored.

KEYWORDS Greedy algorithm, Dynamic Programming, Algebra of Programming.

RESUMO

O estudo matemático do algoritmo *ganancioso* («greedy») serve como guia para o estudo da programação dinâmica, cujo corpo de conhecimento permanece desorganizado e obscuro a uma grande parte da comunidade de engenharia de software. Este estudo visa estruturar esse corpo de conhecimento, fazendo a ponte entre a abordagem popular baseada em exemplos e os métodos mais teóricos da literatura científica. Para esse efeito, a teoria dos matroides é explorada pelo uso de uma álgebra de relações *pointfree*, e aplicada a problemas «greedy» e de programação dinâmica. Um catálogo de tais problemas é compilado, e é feita uma caraterização geral de algoritmos de programação dinâmica. Em paralelo, é explorada a teoria do combinador relacional de «thinning».

PALAVRAS-CHAVE Algoritmo «greedy», programação dinâmica, álgebra da programação.

CONTENTS

LIST OF FIGURES

Part I

INTRODUCTORY MATERIAL

1

INTRODUCTION

1.1 CONTEXT

Algorithms research is one of the cornerstones of computer science. Although much older than computers themselves, interest in studying algorithms and their structure grew as computers became more powerful and more widespread. Today, there is a vast body of scientific knowledge concerning the nature of algorithms, and a wide market for applying this knowledge. Many researchers focus their study on the structure of algorithms and their characterization. Examples of this type of work are the design and implementation of different algorithmic techniques, the identification of complexity classes for algorithms, or the characterization of different approaches tackling the same problem (e.g. parsing).

Dynamic Programming (DP) is an important area of work in algorithms research. The term, originally coined by Richard Bellman in 1952 to refer to a mathematical optimization method for solving multi-stage decision problems, is nowadays more commonly used in computer programming as an umbrella term for a collection of techniques to solve certain types of recursive problems (Bellman, 1952, 1954, 1963, 2010).

As is to be expected, these two meanings are not divorced from each other. Richard Bellman's *Principle of Optimality* — stating that present decisions can be made with the assumption that future decisions follow an optimal policy — correlates directly to a fundamental characteristic of DP problems: the presence of an *optimal substructure*. An optimal substructure for a problem is one from which an optimal solution can be derived from the optimal solution to its subproblems. If multiple subproblems rely on the same results, that is, if an even smaller subproblem needs to be solved in order to solve both, then these are said to be *overlapping subproblems*. As a programming discipline, Dynamic Programming is used to avoid repeated calculations of overlapping subproblems, and this is an essential characteristic of the discipline.

Beyond this simple characterization of DP *problems*, there is also a characterization of DP *techniques*. A common approach is to divide algorithms according to the type of strategy they use — either a 'top-down' or a 'bottom-up' strategy (Cormen et al., 2009; Bird and Gibbons, 2020). The top-down strategy solves problems recursively, considering larger problems first, and then their subproblems. 'Memoization' fits this description, as it caches solutions to subproblems upon their first completed calculation, which are then requested by a recursive call in a larger problem. The bottom-up strategy starts from the solution to the smallest problems, and uses them to build solutions to larger problems. For instance, the 'tabulation' technique stores all previously

calculated solutions in a table, trading space needed to store calculated solutions for the time needed to repeat their calculation.

An alternative way to conceptualize Dynamic Programming is in relation to brute force algorithms and the so-called *greedy algorithm*. The distinction, which is more relevant to combinatorial optimization problems, is in the number of candidate solutions considered. In the brute force approach, every single candidate solution must be considered. The greedy algorithm, on the other hand, only requires keeping track of a single solution. Dynamic Programming is a middle ground between the two: the solution space can be culled by finding partial solutions guaranteed not to result in the optimal solution (Bird and de Moor, 1997).

This description fits nicely with the way combinatorial optimization problems are typically specified, and their solutions derived. Firstly, the solution space is defined as encompassing all feasible solutions. Then, an optimization criterion is given, indicating which are the optimal solutions. For DP problems, the derivation consists in showing that an optimal solution is still obtained even when a portion of the solution space is excluded from consideration. For the greedy algorithm, it must show that disregarding all but one solution at each decision point will still yield the optimal one.

Well-known problems such as, for instance, calculating Fibonacci numbers are DP problems: they can be specified by a recurrence relation and thus exhibit optimal substructure, and this recurrence relation makes subproblems overlap. However, these are not optimization problems, and so the conceptualization involving solution spaces and optimization criteria cannot be straightforwardly applied. Moreover, by the nature of their recurrence relation, these problems can potentially be solved by techniques, such as tupling (Pettorossi, 1984), that seem to stray away from the intuitive notion of Dynamic Programming as considering multiple candidate solutions, and culling a solution space.

Problems (or families of problems) exist that can be solved by the greedy algorithm or call for a DP approach depending on the details of their formulation. For instance, the coin changing problem — making change with the fewest coins possible — admits a greedy algorithm (for every possible input) only for 'canonical' coin systems, while for others this algorithm fails (Cai, 2009). In fact, the previous statement is tautological: the notion of canonicity for coin systems is defined exactly by whether or not the greedy algorithm yields the optimal solution.

The greedy algorithm has been extensively studied through the lens of *matroid theory* (Welsh and Society, 1976; Korte and Lovász, 1984; Helman et al., 1995; Vince, 2002). Matroid theory concerns itself with studying matroidal structures in different areas of mathematics, such graph theory, lattice theory, and combinatorics, establishing links and generalizations among different mathematical fields. The greedy algorithm is an algorithmic property of a matroid, for suitable objective functions (Welsh and Society, 1976). Korte and Lovász (1984) relax the definition of a matroid to obtain *greedoids* and give sufficient conditions for the correctness of the greedy algorithm under a greedoid. Helman et al. (1995) use *matroid embeddings* to prove the correctness of the greedy algorithm for linear and bottleneck objective functions under certain set systems, and to characterize different problems according to independence properties in terms of matroid theory. Vince (2002) gives a framework based on a slightly different notion of a set system to prove the greedy algorithm for some linear weight functions.

Although mechanically similar, applications of the greedy algorithm for different problems are often justified through different means. Curtis (2003) proposes a five-category classification for greedy algorithms, based on four principles: 'Best-Global', 'Better-Global', 'Best-Local' and 'Better-Local'. These principles describe how candidate solutions constructed from different choices relate to each other. For instance, the Better-Global principle states that making a better choice always results in a better solution when the algorithm terminates. This is not true for every problem for which the greedy algorithm gives the optimal solution, because only the Best-Global principle — the best choice always results in a better solution upon termination — is true in every case.

In contrast, the study of Dynamic Programming tends to be example-based, especially in the popular literature. The process of designing algorithms often requires 'eureka' steps that may be hard to envision for someone not experienced enough. For instance, tabulation methods need to determine appropriate dimensions for the table of cached solutions, and there isn't a unified approach for this. It is even possible to come up with different tabulation schemes for the same problem, see e.g. (Boiten, 1992). Each problem comes with unique characteristics that must be exploited to yield an efficient algorithm.

Nevertheless, there is a vast body of research which concerns itself with finding techniques and general strategies to solve these problems. These may lead to the creation and adoption of mathematical tools to help designing and implementing software systems which rely on DP algorithms.

A functional approach to data structures provides the necessary basic building blocks for constructing efficient functional algorithms. Okasaki (2004) uses *numeral representations* as analogues for representations of container structures, and *structural decomposition* and *structural abstraction* to derive implementations of functional (persistent) data structures, guaranteeing better asymptotic complexity of common operations.

In the area of program transformation, there are some strategies to transform recursive, easy-to-prove-correct algorithms for DP problems into equivalent and more efficient programs. Boiten (1992) describes a strategy of "inverting the order of evaluation" as a means to derive efficient linear recursive and iterative functions using tabulations. Pettorossi (1984) uses a "tupling" strategy to transform general recursive programs to linear recursive programs, and ultimately iterative ones using other techniques.

In their "Algebra of Programming" (AoP) textbook, Bird and de Moor (1997) lay out an algebraic framework based on relation algebra and use it to derive the design and implementation of DP algorithms for optimization problems from their relational specification, conditioning their correctness on a set of appropriate conditions generated by the process. The concept of *thinning*, introduced in the AoP textbook, gives rise to "thinning algorithms", which are characterized by the culling of the solution space to disregard partial solutions that are not useful to solving the overall problem. Morihata et al. (2014) pair thinning with *incrementalization* — a technique used to improve efficiency by re-using previously computed partial solutions (Liu and Stoller, 1999) — to describe the processes behind DP algorithms.

Using this framework, some case studies have been developed. Bird et al. (2000) and Mu (2000) derive linear time algorithms to the minimum height tree problem. Mu et al. (2010) generalize the thinning theorem of Bird and de Moor (1997) to derive solutions for fully polynomial-time approximation scheme (FPTAS) algorithms in a datatype-generic way. Mu and Oliveira (2012) added to this approach by devising a new combinator to express

optimization processes in a relation-algebraic way. *Metaphorisms*, which express relationships between input and output through shared observed properties, have been used to specify and solve some problems using this framework, including recursive optimization problems (Oliveira, 2018).

Other parts of the literature also exploring optimization problems solved by the greedy algorithm and Dynamic Programming use simpler frameworks to increase accessibility. For instance, Bird and Rabe (2019) make use of "nondeterministic functions", which produce multiple values, and give a theoretical framework with which to reason about them, while Bird and Gibbons (2020) prefer the even simpler approach of using only pure functions to specify and reason about problems.

1.2 APPROACH

The history of software engineering is marked by increasing levels of abstraction in the way computer systems are programmed. Starting from the manual input of machine code, programmers soon found ways to make their jobs easier with assemblers, linkers, and ultimately compilers and interpreters for high level languages. A number of programming paradigms emerged from the flurry of new programming languages being created in the decades following the invention of modern electronic computers. Among them, the imperative and object-oriented paradigms proved most influential, through languages such as *C* and *Java*. Their overarching goal was to provide the programmer with tools to better express how a software system should work. This meant giving the programmer the option of organizing the program's code in modules and/or classes, in effect detailing the different components of a system's architecture, and how they should interact with each other.

Parallel to this trend, and with the realization of the link between mathematical proofs and computer programs, termed the Curry-Howard correspondence, there was an effort to make software design more of an engineering discipline, using scientific principles to design and implement large software systems. In the programming language space, two important paradigms, functional programming and logic programming, were developed, with the end result being a more declarative style of programming: instead of detailing *how* a system should work, ideally, a programmer would only specify *what* it should do (Cohen, 1995).

Although less popular than imperative and object-oriented languages, functional programming languages are widely used in research circles, and, in turn, have seen the greatest benefits from the more mathematical approach, aiming for a greater sense of generality with features such as first-class (higher-order) functions, type polymorphism, algebraic datatypes, among others (Hudak, 1989). More recently, some of these features have even made their way into other, non-functional languages, for instance pattern matching in Python 3.10 (Kohn and van Rossum, 2020).

In some areas of software engineering, namely those concerned with the design of safety-critical computer systems, the use of *formal methods* in software (and hardware) engineering is almost mandatory. As small mistakes can make a catastrophic difference, the cost of ensuring dependability more than makes up for the potential risk of operating a faulty system. Notable examples of the use of formal methods in industry setting include aircraft manufacturing and maintenance (Cofer, 2012), hardware design (Kern and Greenstreet, 1999), and the design and implementation of cryptographic protocols (Meadows, 1995).

Engineers in these industries make use of formal methods tools developed by computer scientists based on prior theoretical work. Among others, there are now tools that allow for formal verification of programs against specifications, and even automatic program synthesis from specifications. However, many of these tools are simply too complicated and involve too many technicalities to be of use for a large number of regular programmers. It is often the case that, even for relatively simple programs, the user is tasked with proving trivial properties through somewhat arcane methods, as the system is unable to automatically discharge their proof.

Moreover, these technicalities prevent programmers from understanding how programs work, when viewed from outside. In fact, the usefulness of formal methods in software engineering comes not only from proving programs correct and building correct programs, but also from understanding the reasons why indeed those programs are correct.

In this setting, the work described in this dissertation will adopt the AoP mathematical framework. The power and flexibility of relational algebra facilitates problem specification, and allows for a general exploration of recursive structures in Greedy and Dynamic Programming problems. Another advantage of this framework is that nondeterminism is automatically encoded within relations themselves. So, many derivations can be divided in two parts: one where the essential concepts and properties of the problem are applied, and the other which focuses on obtaining an executable function by eliminating nondeterminism or by directly implementing it.

Furthermore, the connection between matroid theory, the greedy algorithm, and dynamic programming will be explored using the calculational approach of AoP. From problem specifications, structural properties will be identified, encoded in relational algebra, and then used in to derive implementations.

In short, the main goal of this project is to expand the extent to which formal methods, specifically program synthesis from a specification using the calculational style of AoP, can be used to derive Greedy and Dynamic Programming algorithms, while aiming for generality and ease of understanding.

1.3 STRUCTURE OF THE DISSERTATION

Chapter 2 will cover the required relation algebra background, basic relation algebra, the notions of shrinking, catamorphism, hylomorphism, and so on. It ends by presenting the main theorems related to Dynamic Programming. Chapter 3 goes over several important inductive relations operating on lists. Relations for permutations, subsequences and sublists are introduced, along with associated results and rules for monotonicity. Chapter 4 explores the algorithmic definition of the greedy algorithm through matroid theory. Chapter 5 expands on the theory underlying the thinning relational operator, including preorder thinning and recursive thinning. Chapter 6 gives a characterization of Dynamic Programming algorithms and develops a couple of case studies that put the contributions of the previous chapter into practice.

Appendix A gives a selection of problems in the greedy and DP problem space. To preserve the fluitity of the main text, some proofs are deferred to Appendix B.

2

BACKGROUND

This chapter will first cover the basic constructs and combinators of relational algebra, followed by inductive (recursive) relations and their laws. Finally, it lays the groundwork for dealing with optimization problems by introducing the *shrinking* operator (Mu and Oliveira, 2012), reviewing theorems used by Bird and de Moor (1997), and introducing operators to handle nondeterminism.

2.1 BASIC RELATION ALGEBRA

In functional programming, the most important objects are functions. They take an input and produce a single output. In relation calculus, we generalize this to binary relations, which may produce multiple outputs. As we will shortly see, this means that functions are simple relations.

Composing functions is a common and useful operation. Relational composition $(R \cdot S)$ takes into account the multiple values that may be produced by S. It is defined as follows:

$$B \xleftarrow{R} A \xleftarrow{S} C \qquad b(R \cdot S)c \;\equiv\; \langle \exists\, a \;:\; b\, R\, a \;:\; a\, S\, c \rangle \tag{2.1}$$

In English, an element c is related to an element b by $(R \cdot S)$ if (and only if) there exists an element a related to b by R, and such that c is related to it by S. This compact notation for relational composition allows us to deal with existential quantification in a pointfree style in our proofs.

Unlike functions, which do not necessarily have corresponding converse functions, *every* relation $B \xleftarrow{R} A$ has a converse $A \xleftarrow{R^\circ} B$ defined by:

$$b\, R\, a \;\Leftrightarrow\; a\, R^\circ\, b \tag{2.2}$$

Naturally, the converse of a converse is the relation itself — converse is an involution

$$(R^\circ)^\circ = R \tag{2.3}$$

and commutes with composition in a contravariant way:

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{2.4}$$

The following construct allowing the relation of different properties, given by the functions f and g, of two different elements is useful when dealing with more complex types:

$$b\ (f^\circ \cdot R \cdot g)\ a \ \Leftrightarrow\ (f\ a)\ R\ (g\ b) \tag{2.5}$$

It is particularly useful for the purposes of our work when $f = g$, so the abbreviation

$$R_f = f^\circ \cdot R \cdot f \tag{2.6}$$

serves to make notation more compact. For example, $(\leqslant)_{\mathsf{length}}$ compares the length of two lists. While function equality can be done by extensionality

$$f = g \ \equiv\ \langle \forall\ a\ :\ a \in A :\ f\ a = g\ a \rangle$$

where A is the domain of f and g, there are two methods, besides direct equality, to prove relation equality: *circular inclusion*, and *indirect equality*. Both of these rely on the notion of relation inclusion, defined as follows:

$$R\ \subseteq\ S\ \equiv\ \langle \forall\ a, b\ ::\ a\ R\ b \Rightarrow a\ S\ b \rangle \tag{2.7}$$

Circular inclusion arrives at equality by proving, in both directions, that one relation is smaller than, or, at most, equal to the other. This method is often called "ping-pong", and in proofs of this type the two steps will be referred to as the "ping" step and the "pong" step.

$$R = S\ \equiv\ R \subseteq S \wedge S \subseteq R \tag{2.8}$$

In indirect equality, we prove that inclusion under one relation is equivalent to inclusion under the other. Alternatively, we prove that all relations having R as a subset have S as a subset, and vice-versa.

$$R = S \quad \equiv \quad \langle \forall\ X\ ::\ (X\ \subseteq\ R\ \Leftrightarrow\ X\ \subseteq\ S) \rangle \tag{2.9}$$

$$\equiv \quad \langle \forall\ X\ ::\ (R\ \subseteq\ X\ \Leftrightarrow\ S\ \subseteq\ X) \rangle \tag{2.10}$$

Binary relations can be viewed as sets of pairs. As such, the notions of intersection and union exist in this universe. The operators *meet* and *join* are defined (pointwise) as such:

$$a\ (R \cap S)\ b \ \Leftrightarrow\ a\ R\ b \wedge a\ S\ b \tag{2.11}$$

$$a\ (R \cup S)\ b \ \Leftrightarrow\ a\ R\ b \vee a\ S\ b \tag{2.12}$$

Using relational inclusion, we can state the universal properties of meet and join in pointfree form:

$$X \subseteq R \cap S\ \equiv\ X \subseteq R\ \wedge\ X \subseteq S \tag{2.13}$$

$$R \cup S \subseteq X\ \equiv\ R \subseteq X\ \wedge\ S \subseteq X \tag{2.14}$$

On occasion, it is useful to deal with relational difference ($-$), which interacts with the meet and join operators

$$R \subseteq S \cup T \quad \Leftrightarrow \quad R - T \subseteq S \tag{2.15}$$
$$R - S = R \cap \neg\, S \tag{2.16}$$

where $\neg$ is the relational negation operator:

$$\neg\, R = \top - R \tag{2.17}$$

As can be easily checked, converse is monotonous with meet and join:

$$(R \cap S)^\circ = R^\circ \cap S^\circ \tag{2.18}$$
$$(R \cup S)^\circ = R^\circ \cup S^\circ \tag{2.19}$$

Finally, bilinearity of relation composition with respect to relational join

$$R \cdot (S \cup T) \;=\; (R \cdot S) \cup (R \cdot T) \tag{2.20}$$
$$(S \cup T) \cdot R \;=\; (S \cdot R) \cup (T \cdot R) \tag{2.21}$$

as well as left semi-linearity with respect to relational meet

$$R \cdot (S \cap T) \subseteq (R \cdot S) \cap (R \cdot T) \tag{2.22}$$
$$(S \cap T) \cdot R \subseteq (S \cdot R) \cap (T \cdot R) \tag{2.23}$$

will be useful for some proofs. To strengthen these to equalities, side conditions are introduced:

$$R \cdot (S \cap T) = (R \cdot S) \cap (R \cdot T) \;\Leftarrow\; \ker R \cdot S \subseteq S \;\vee\; \ker R \cdot T \subseteq T \tag{2.24}$$
$$(S \cap T) \cdot R = (S \cdot R) \cap (T \cdot R) \;\Leftarrow\; S \cdot \mathrm{img}\, R \subseteq S \;\vee\; T \cdot \mathrm{img}\, R \subseteq T \tag{2.25}$$

So far, we have covered a few relational combinators and few ways to prove relation equality. What is missing is the body of laws that governs how equations using these basic combinators can be transformed. In the rest of this section, a selection of laws relevant to our work will be introduced.

The most fundamental properties of inclusion are transitivity

$$R \;\subseteq\; S \wedge S \;\subseteq\; T \;\Rightarrow\; R \;\subseteq\; T \tag{2.26}$$

and monotonicity, which applies, among others, to the basic relational combinators of composition, converse, meet and join:

$$R \;\subseteq\; S \wedge T \;\subseteq\; U \Rightarrow R \cdot T \;\subseteq\; S \cdot U \tag{2.27}$$
$$R \;\subseteq\; S \Rightarrow R^\circ \;\subseteq\; S^\circ \tag{2.28}$$
$$R \;\subseteq\; S \wedge T \;\subseteq\; U \Rightarrow R \cap T \;\subseteq\; S \cap U \tag{2.29}$$

$$R \subseteq S \wedge T \subseteq U \Rightarrow R \cup T \subseteq S \cup U \tag{2.30}$$

Due to transitivity of inclusion (2.26), we are able to prove $R \subseteq S$ by finding a mid-point relation $R \subseteq M \subseteq S$, proving both that $R \subseteq M$ and $M \subseteq S$. Since one these statements will be trivial, assumed or otherwise given by another theorem, we shall use it as justification whenever we wish to reduce the proof to the other statement. So, depending on which we choose, we can "lower the upper side"

$$\begin{array}{ll} & R \subseteq S \\ \Leftarrow & \quad \{ M \subseteq S \} \\ & R \subseteq M \end{array} \tag{2.31}$$

or "raise the lower side":

$$\begin{array}{ll} & R \subseteq S \\ \Leftarrow & \quad \{ R \subseteq M \} \\ & M \subseteq S \end{array} \tag{2.32}$$

Bird and de Moor (1997) derive a *modular law* that combines these relational combinators, like so:

$$R \cdot S \cap T \subseteq R \cdot (S \cap R^\circ \cdot T) \tag{2.33}$$

Per (2.28), we take the converse of each side (and rename) to get:

$$R \cdot S \cap T \subseteq (R \cap T \cdot S^\circ) \cdot S \tag{2.34}$$

These laws allow us to weaken or strengthen a condition by choosing either side as a mid-point between two other relations, and applying (2.31) or (2.32).

To understand what they mean exactly, let us take a look at an example: take aRb to mean "person b has driven a car a", aSb to mean "person b knows person a", and aTb to mean "person b owns a car a". Then (2.34) tells us that if a person c owns a car a, and knows some person b who has driven a, then we can say that person c knows some person b who has driven a, and that person is known by some person d who owns car a. The second statement is always true when the first one is, instantiating d to c to obtain the original.

TAXONOMY OF BINARY RELATIONS Let us now define some basic properties of some relations, which will be the starting point to describe more complex properties.

A relation is *reflexive* if and only if every element is related to itself:

$$R \text{ is reflexive} \quad \equiv \quad id \subseteq R \tag{2.35}$$

A relation is *coreflexive* if and only if it is a subset of the identity relation:

$$R \text{ is coreflexive} \quad \equiv \quad R \subseteq id \tag{2.36}$$

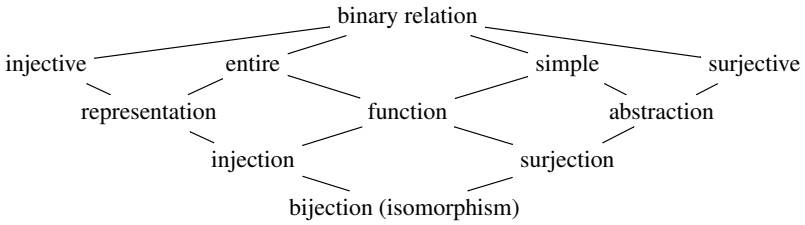


Figure 1.: Binary relation taxonomy

A relation is *transitive* if and only if $c\ R\ a$ whenever $c\ R\ b$ and $b\ R\ a$:

$$R \text{ is transitive} \quad \equiv \quad R \cdot R \subseteq R \tag{2.37}$$

A *preorder* is a relation that is both reflexive and transitive. One such relation is the commonplace numerical comparison operator $\leqslant$. Many proofs in later sections will rely on this operator's properties as a preorder.

Additionally, a relation is:

- *entire* if it is defined for every element of its domain.
- *injective* if no two elements of its domain are related to the same element.
- *surjective* if, for every element of its range, there is at least one element related to it.
- *simple* if each element of its domain is related to at most one element.

To define these last four properties using relation algebra, we first introduce the notions of *kernel* and *image*

$$\ker R = R^\circ \cdot R \tag{2.38}$$
$$\operatorname{img} R = R \cdot R^\circ \tag{2.39}$$

as the kernel of R relates inputs that share outputs, and its image relates outputs that share inputs.

$$\text{R injective:} \quad \ker R \subseteq id \tag{2.40}$$
$$\text{R simple:} \quad \operatorname{img} R \subseteq id \tag{2.41}$$
$$\text{R entire:} \quad id \subseteq \ker R \tag{2.42}$$
$$\text{R surjective:} \quad id \subseteq \operatorname{img} R \tag{2.43}$$

Figure 1 shows a taxonomy of binary relations based on the combination of the previous four properties. The properties of different kinds of relations have dualities under converse, that is:

$$R \text{ is a bijection} \quad \equiv \quad R \text{ and } R^\circ \text{ are both functions} \tag{2.44}$$
$$R \text{ is injective} \quad \equiv \quad R^\circ \text{ is simple} \tag{2.45}$$
$$R \text{ is surjective} \quad \equiv \quad R^\circ \text{ is entire} \tag{2.46}$$

By extension, the converse of a representation is an abstraction, the converse of an injection is a surjection, and vice-versa.

Due to transitivity of inclusion (2.26), and the fact that a larger relation also has a larger (or at least equal) kernel and image, making a relation bigger or smaller can preserve some of its properties, as follows:

$$S \text{ is injective (simple)} \quad \Leftarrow \quad S \subseteq R \text{ and } R \text{ is injective (simple)} \tag{2.47}$$

$$S \text{ is entire (surjective)} \quad \Leftarrow \quad R \subseteq S \text{ and } R \text{ is entire (surjective)} \tag{2.48}$$

TOP AND BOTTOM The $\top$ ("top") and $\bot$ ("bottom") relations

$$b \top a \quad \equiv \quad true \tag{2.49}$$

$$b \bot a \quad \equiv \quad false \tag{2.50}$$

are the upper and lower bound for all relations of a certain relational type, meaning that, for any relation $A \xleftarrow{R} B$:

$$\bot \subseteq R \subseteq \top \tag{2.51}$$

(Note that *true* and *false* in the definitions above are logical constants, while in the rest of the text, they are constant functions returning the values *True* and *False* respectively.)

These relations are the zero and identity elements of the relational join and meet operations:

$$R \cup \bot = R \tag{2.52}$$

$$R \cap \bot = \bot \tag{2.53}$$

$$R \cup \top = \top \tag{2.54}$$

$$R \cap \top = R \tag{2.55}$$

If a relation is entire, then it is absorbed by $\top$ when composed to the right of it:

$$\top \cdot R = \top \quad \Leftarrow \quad R \textit{ is entire} \tag{2.56}$$

Since $\top$ is larger than any relation (2.51), it suffices to prove $\top \subseteq \top \cdot R$:

$$\begin{array}{ll} & \top \subseteq \top \cdot R \\ \Leftarrow & \quad \{ \text{ } R \text{ is assumed entire (2.42), raising the lower side (2.32) } \} \\ & \top \cdot R^\circ \cdot R \subseteq \top \cdot R \\ \Leftarrow & \quad \{ \text{ monotonicity (2.27) } \} \\ & \top \cdot R^\circ \subseteq \top \\ \equiv & \quad \{ \text{ } \top \text{ above everything (2.51) } \} \\ & true \end{array}$$

□

On the other hand, and by the very definition of relational composition, we have:

$$\bot \cdot R = R \cdot \bot = \bot \tag{2.57}$$

Let us lay out the precedence of operations for relation algebra (plus logical operators used to write theorems)

- Unary operators bind tighter than binary combinators;
- The converse unary operator binds tighter than the power transpose operator — ΛS° is parsed as $\Lambda(S^\circ)$;
- Composition $M \cdot N$ binds tighter than all other binary combinators, e.g. $S \cdot T \upharpoonright R$ is parsed as $(S \cdot T) \upharpoonright R$;
- Combinators represented by symbols that are taken from other fields of mathematics are often chosen because they share similar properties, and so take precedence over related operators in a similar way, for example, multiplication and addition, so that the type specification $A + A \times B$ is parsed as $A + (A \times B)$;
- The division ($/$ and $\backslash$), shrinking ($\upharpoonright$) and thinning ($\downharpoonleft$) combinators take higher precedence than meet and join (resp. $\cap$ and $\cup$), and the operators mentioned in the previous point;
- Meet and join take higher precedence than relation equality and inclusion (resp. $=$ and $\subseteq$);
- Logical operators ($\wedge$, $\Rightarrow$, $\equiv$, etc.) bind less tightly than all others above, and take precedence over each other in the conventional way.
- Underbraces ($\underbrace{\quad}$), besides giving a name to part of an expression, also act as a grouping symbol, so $\underbrace{R \cup S \cdot T}_{X} \cdot U$ should be treated as $(R \cup S \cdot T) \cdot U$.

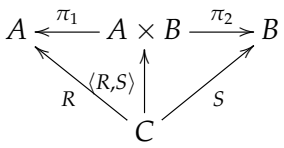
When dealing with functions, we can apply the 'shunting' laws — a function and its converse can be exchanged between the two sides of an inclusion as follows:

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S \tag{2.58}$$

$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f \tag{2.59}$$

Next, we cover some basic algebraic datatypes and their associated operations.

RELATIONAL PAIRINGS Relational pairing combines two relations $A \xleftarrow{R} C$ and $B \xleftarrow{S} C$ to form a relation $\langle R, S \rangle$ ("split of R and S"), as shown in the following type diagram:



This construct stipulates that both composite relations must have the same domain type, and its range is a subset of the Cartesian product of both range types. As the type diagram suggests, the following universal property holds:

$$X \subseteq \langle R, S \rangle \equiv \begin{cases} \pi_1 \cdot X \subseteq R \\ \pi_2 \cdot X \subseteq S \end{cases} \tag{2.60}$$

By shunting (2.58) and the universal property of meet (2.13), we have a closed-form definition of relational pairing:

$$\langle R, S \rangle = \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S \tag{2.61}$$

From the universal law we also get the cancellation laws

$$\pi_1 \cdot \langle R, S \rangle \subseteq R \tag{2.62}$$
$$\pi_2 \cdot \langle R, S \rangle \subseteq S \tag{2.63}$$

which are strengthened to equalities when the relation from the opposite side is entire:

$$\pi_1 \cdot \langle R, S \rangle = R \Leftarrow S \text{ is entire} \tag{2.64}$$
$$\pi_2 \cdot \langle R, S \rangle = S \Leftarrow R \text{ is entire} \tag{2.65}$$

The following law allows us to fuse the converse of a pairing and a pairing into a relational intersection involving their projections:

$$\langle R, S \rangle^\circ \cdot \langle X, Y \rangle = R^\circ \cdot X \cap S^\circ \cdot Y \tag{2.66}$$

The projections π_1 and π_2 can be defined as the converse of a pairing, which will also be useful in some proofs:

$$\pi_1 = \langle id, \top \rangle^\circ \tag{2.67}$$
$$\pi_2 = \langle \top, id \rangle^\circ \tag{2.68}$$

The fusion law

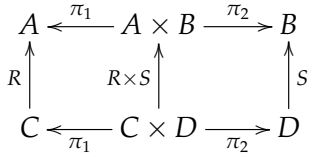
$$\langle R, S \rangle \cdot T = \langle R \cdot T, S \cdot T \rangle \Leftarrow R \cdot \mathrm{img}\, T \subseteq R \vee S \cdot \mathrm{img}\, T \subseteq S \tag{2.69}$$

also holds if T is simple.

KRONECKER PRODUCTS FOR RELATIONS In the category of relations, a Kronecker product can be defined as a specific pairing:

$$R \times S = \langle R \cdot \pi_1, S \cdot \pi_2 \rangle \tag{2.70}$$

Now, with relations $A \xleftarrow{R} C$ and $B \xleftarrow{S} D$, we have the type diagram



showing how R and S operate on the two sides of the product. The Kronecker product interacts with relational pairing through the following absorption law:

$$(R \times S) \cdot \langle P, Q \rangle = \langle R \cdot P, S \cdot Q \rangle \tag{2.71}$$

The free theorems of π_1 and π_2, proven as instances of Reynolds abstraction theorem in (Barbosa et al., 2008), state that:

$$\pi_1 \cdot (R \times S) \subseteq R \cdot \pi_1 \tag{2.72}$$

$$\pi_2 \cdot (R \times S) \subseteq S \cdot \pi_2 \tag{2.73}$$

Contrary to what it might seem, R and S do not act completely independently of each other, so stating these laws as equalities is not possible without imposing restrictions. The problem arises when a pair contains an element outside a relation's domain, and another that is part of opposite relation's domain. To avoid this situation, we stipulate that the excluded relations are entire.

$$\pi_1 \cdot (R \times S) = R \cdot \pi_1 \Leftarrow S \textit{ is entire} \tag{2.74}$$

$$\pi_2 \cdot (R \times S) = S \cdot \pi_2 \Leftarrow R \textit{ is entire} \tag{2.75}$$

Proof of (2.74):

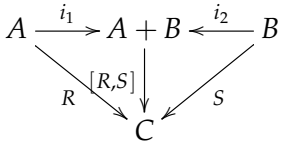
$$\begin{aligned}
& \pi_1 \cdot (R \times S) \\
= & \quad \{ \ \pi_1 = \langle id, \top \rangle^\circ \text{ (2.67), Kronecker product definition (2.70) } \} \\
& \langle id, \top \rangle^\circ \cdot \langle R \cdot \pi_1, S \cdot \pi_2 \rangle \\
= & \quad \{ \ \text{pairings (2.66), } \top^\circ = \top \text{ by (2.2) } \} \\
& R \cdot \pi_1 \cap \top \cdot S \cdot \pi_2 \\
= & \quad \{ \ \top \cdot S \cdot \pi_2 = \top \text{ because } S \cdot \pi_2 \text{ is entire (2.56) } \} \\
& R \cdot \pi_1 \cap \top \\
= & \quad \{ \ \top \text{ is identity element of meet (2.55) } \} \\
& R \cdot \pi_1
\end{aligned}$$

□

A similar reasoning applies symmetrically to the proof of (2.75).

RELATIONAL BIPRODUCTS Relational products and coproducts are dual constructs in the category of relations, meaning that they have the same structure, only the direction of the arrows is reversed.

A relational *coproduct* is a relation that produces a sum type $A + B$ from relations $C \xleftarrow{R} A$ and $C \xleftarrow{S} B$, and contains either a value of type A or B, accompanied by a tag. $A + B$ represents the disjoint union of types A and B. Its type structure is shown in the following diagram:



where i_1 and i_2 are the injections, which can be defined as:

$$i_1 = [id, \bot] \tag{2.76}$$

$$i_2 = [\bot, id] \tag{2.77}$$

Again, the diagram suggests the universal property:

$$X = [R, S] \equiv \begin{cases} X \cdot i_1 = R \\ X \cdot i_2 = S \end{cases} \tag{2.78}$$

Note the difference to the universal property for pairings (2.60). Relational inclusion in pairing cannot be tightened to equality, while relaxing the universal property for coproducts is possible:

$$X \subseteq [R, S] \equiv \begin{cases} X \cdot i_1 \subseteq R \\ X \cdot i_2 \subseteq S \end{cases} \tag{2.79}$$

If we make $X := [Y, Z]$ in either (2.78) or (2.79), we obtain laws for coproduct equality

$$[Y, Z] = [R, S] \equiv \begin{cases} Y = R \\ Z = S \end{cases} \tag{2.80}$$

and relational inclusion:

$$[Y, Z] \subseteq [R, S] \equiv \begin{cases} Y \subseteq R \\ Z \subseteq S \end{cases} \tag{2.81}$$

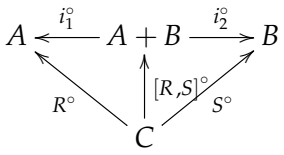
We can also define coproducts in closed form:

$$[R, S] = R \cdot i_1^\circ \cup S \cdot i_2^\circ \tag{2.82}$$

Because a coproduct is a join of two relations, it obeys a fusion law:

$$R \cdot [S, T] = [R \cdot S, R \cdot T] \tag{2.83}$$

The dual construct to the relational coproduct, the relational product, can be obtained by inverting the direction of the arrows in the type diagram:



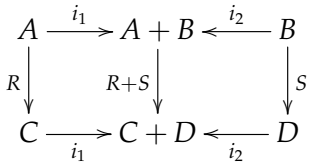
making i_1° and i_2° projections from the sum type. Since $[R,S]^\circ = [R^\circ, S^\circ]$ by virtue of (2.82) and (2.19), we also have the closed form definition of a product:

$$[R,S]^\circ = i_1 \cdot R^\circ \cup i_2 \cdot S^\circ \tag{2.84}$$

The following law simplifies composition between a coproduct and a product:

$$[R,S] \cdot [T,U]^\circ \;=\; R \cdot T^\circ \cup S \cdot U^\circ \tag{2.85}$$

RELATIONAL DIRECT SUM As relational pairings give rise to the Kronecker product, a similar strategy gives us the relation direct sum $R + S$. With relations $C \xleftarrow{R} A$ and $D \xleftarrow{S} B$, we have:



The relational direct sum can then be defined as a co-product:

$$R + S \;=\; [i_1 \cdot R, i_2 \cdot S] \tag{2.86}$$

Essentially, this allows us to have different computational "tracks" for different types. The following absorption law

$$[S,Q] \cdot (P + R) \;=\; [S \cdot P, Q \cdot R] \tag{2.87}$$

allows us to extend the tracks and absorbs them into the coproduct. Naturally, one can also fuse two direct sums into one

$$(S + Q) \cdot (P + R) \;=\; S \cdot P + Q \cdot R \tag{2.88}$$

and proving inclusion entails proving the inclusion on both sides separately:

$$S + Q \subseteq P + R \;\equiv\; \begin{cases} S \subseteq P \\ Q \subseteq R \end{cases} \tag{2.89}$$

It will be convenient to apply laws (2.87), (2.83) and (2.81) in a single step ("coproducts"), introducing factors id or $id + id$ where necessary:

$$P \cdot [R, S] \cdot (W + X) \subseteq Q \cdot [T, U] \cdot (Y + Z) \quad \equiv \quad \begin{cases} P \cdot R \cdot W \subseteq Q \cdot T \cdot Y \\ P \cdot S \cdot X \subseteq Q \cdot U \cdot Z \end{cases} \tag{2.90}$$

RELATIONAL DIVISIONS Relational composition allows us to express existential quantification in a pointfree style. Relational division, in turn, substitutes universal quantification in relation calculus:

$$\begin{aligned} c\ (R \mathbin{/} S)\ a \quad &\Leftrightarrow \quad \langle \forall\ b\ :\ a\ S\ b :\ c\ R\ b \rangle \\ c\ (S \setminus R)\ a \quad &\Leftrightarrow \quad \langle \forall\ b\ :\ b\ S\ a :\ b\ R\ c \rangle \end{aligned}$$

Left division $R \setminus S$ (read "R under S") and right division $R \mathbin{/} S$ ("R over S") can also be defined by the Galois connections

$$R \cdot X \subseteq S \quad \Leftrightarrow \quad X \subseteq R \setminus S \tag{2.91}$$

$$X \cdot R \subseteq S \quad \Leftrightarrow \quad X \subseteq S \mathbin{/} R \tag{2.92}$$

which prove extremely useful for the purposes of our work. Taking X as the divisions in their respective Galois connection yields the two cancellation laws:

$$R \cdot (R \setminus S) \subseteq S \tag{2.93}$$

$$(S \mathbin{/} R) \cdot R \subseteq S \tag{2.94}$$

One difficulty that typically arises when dealing with equations involving divisions is the fact that while division on the right hand side can always be converted into composition on the left hand side, the opposite is not true. The following rules will help us overcome this in some cases:

$$R \setminus S \subseteq R^\circ \cdot S \ \Leftarrow\ R \text{ is entire} \tag{2.95}$$

$$S \mathbin{/} R \subseteq S \cdot R^\circ \ \Leftarrow\ R \text{ is surjective} \tag{2.96}$$

The proof of (2.95) is almost immediate (that of (2.96) is similar):

$$\begin{aligned} & R \setminus S \subseteq R^\circ \cdot S \\ \Leftarrow \quad & \{ \text{ cancellation (2.93) and monotonicity (2.31) } \} \\ & R \setminus S \subseteq R^\circ \cdot R \cdot (R \setminus S) \\ \Leftarrow \quad & \{ \text{ monotonicity (2.27) } \} \\ & id \subseteq R^\circ \cdot R \end{aligned}$$

$\square$

The following equalities involving divisions are useful (Oliveira, 2019):

$$R \cdot f = R / f^\circ \tag{2.97}$$
$$f \setminus R = f^\circ \cdot R \tag{2.98}$$
$$R / \bot = \top \tag{2.99}$$
$$R / id = R \tag{2.100}$$
$$(R \setminus S) \cdot f = R \setminus (S \cdot f) \tag{2.101}$$
$$f^\circ \cdot (R \,/\, S) = f^\circ \cdot R \,/\, S \tag{2.102}$$
$$R \,/\, f^\circ \cdot S = (R \,/\, S) \cdot f \tag{2.103}$$
$$R \setminus (f^\circ \cdot S) = f \cdot R \setminus S \tag{2.104}$$
$$R \setminus \top \cdot S = \,!\cdot R \setminus \,!\cdot S \tag{2.105}$$
$$R \,/\, (S \cup P) = R \,/\, S \cap R \,/\, P \tag{2.106}$$
$$(R \setminus S)^\circ = S^\circ \,/\, R^\circ \tag{2.107}$$

All of these have short proofs, mostly reasoning by indirect equality. For example, the proof of (2.102) is laid out:

$$\begin{array}{ll} & X \subseteq f^\circ \cdot (R \,/\, S) \\ \equiv & \quad \{ \text{ shunting (2.59) } \} \\ & f \cdot X \subseteq R \,/\, S \\ \equiv & \quad \{ \text{ right division (2.108) } \} \\ & f \cdot X \cdot S \subseteq R \\ \equiv & \quad \{ \text{ shunting (2.59) } \} \\ & X \cdot S \subseteq f^\circ \cdot R \\ \equiv & \quad \{ \text{ right division (2.108) } \} \\ & X \subseteq f^\circ \cdot R \,/\, S \\ :: & \quad \{ \text{ indirect equality (2.9) } \} \\ & f^\circ \cdot (R \,/\, S) = f^\circ \cdot R \,/\, S \\ \Box & \end{array}$$

Finally, it will be useful to work with divisions $R \setminus R$ and $R \,/\, R$ for some relation $A \xleftarrow{R} A$. Any such division is necessarily a preorder.

$$R \setminus R \text{ is a preorder} \tag{2.108}$$
$$R \,/\, R \text{ is a preorder} \tag{2.109}$$

Proof of (2.108):

$$
\begin{array}{ll}
& R \setminus R \text{ is a preorder} \\
\equiv & \quad \{ \text{ definition of a preorder } \} \\
& \left\{ \begin{array}{l} id \subseteq R \setminus R \\ (R \setminus R) \cdot (R \setminus R) \subseteq R \setminus R \end{array} \right. \\
\equiv & \quad \{ \text{ left division (2.91) } \} \\
& \left\{ \begin{array}{l} R \subseteq R \\ R \cdot (R \setminus R) \cdot (R \setminus R) \subseteq R \end{array} \right. \\
\Leftarrow & \quad \{ \text{ left division cancellation (2.93) } \} \\
& R \cdot (R \setminus R) \subseteq R \\
\Leftarrow & \quad \{ \text{ left division cancellation (2.93) } \} \\
& R \subseteq R \\
& \Box
\end{array}
$$

The proof of (2.109) is similar, using right division laws instead.

SYMMETRIC DIVISION Following Freyd and Scedrov (1990), define $\frac{S}{R}$ as:

$$\frac{S}{R} = R \setminus S \cap R^\circ \,/\, S^\circ \qquad \begin{array}{ccc} B & \xleftarrow{\frac{S}{R}} & C \\ & {\scriptstyle R} \searrow \quad \swarrow {\scriptstyle S} & \\ & A & \end{array} \tag{2.110}$$

Clearly:

$$b \,\frac{S}{R}\, c \;\equiv\; \left\{ \begin{array}{l} \langle \forall\, a \,:\, a\, R\, b :\, a\, S\, c \rangle \\ \langle \forall\, a \,:\, a\, S\, b :\, a\, R\, c \rangle \end{array} \right. \equiv \langle \forall\, a \,::\, a\, R\, c \;\Leftrightarrow\; a\, S\, b \rangle \tag{2.111}$$

One may easily verify other properties such as e.g. (Freyd and Scedrov, 1990):

$$\frac{f}{g} = g^\circ \cdot f \tag{2.112}$$

$$\left(\frac{S}{R}\right)^\circ = \frac{R}{S} \tag{2.113}$$

$$\left\{ \begin{array}{l} \frac{\perp}{R} = R \setminus \perp \\ \frac{R}{\perp} = \perp \,/\, R^\circ \end{array} \right. \tag{2.114}$$

$$\frac{S}{R} \cdot \frac{Q}{S} \subseteq \frac{Q}{R} \tag{2.115}$$

$$f^\circ \cdot \frac{S}{R} \cdot g = \frac{S \cdot g}{R \cdot f} \tag{2.116}$$

$$\frac{f}{id} = f \wedge \frac{id}{f} = f^\circ \tag{2.117}$$

$$\begin{cases} \frac{\top}{R} = R^\circ \mathbin{/} \top \\ \frac{R}{\top} = \top \setminus R \end{cases} \tag{2.118}$$

Symmetric divisions of the form $\frac{f}{f}$ enjoy some special properties. First of all, they are equivalence relations: from (2.115) we prove transitivity, and using (2.112) and shunting (2.58) it is straightforward to prove reflexivity and symmetry. It is also easy to verify that applying ker and img to $\frac{f}{f}$ results in the same relation:

$$\mathrm{ker}\,\frac{f}{f} = \frac{f}{f} \tag{2.119}$$

$$\mathrm{img}\,\frac{f}{f} = \frac{f}{f} \tag{2.120}$$

A RELATION ON FUNCTIONS The so-called "Reynolds arrow" relational operator establishes a relation on two functions f and g parametric on two other arbitrary relations R and S:

$$f\ (R \leftarrow S)\ g \ \equiv\ f \cdot S \subseteq R \cdot g \qquad \begin{array}{ccc} A & \xleftarrow{S} & B \\ {\scriptstyle f}\downarrow & \subseteq & \downarrow{\scriptstyle g} \\ C & \xleftarrow[R]{} & D \end{array} \tag{2.121}$$

This is a powerful operator that satisfies many properties, for instance (Oliveira, 2019):

$$id \leftarrow id \ =\ id \tag{2.122}$$

$$(R \leftarrow S)^\circ \ =\ R^\circ \leftarrow S^\circ \tag{2.123}$$

$$R \leftarrow S \subseteq V \leftarrow U \ \Leftarrow\ R \subseteq V \wedge U \subseteq S \tag{2.124}$$

$$(R \leftarrow V) \cdot (S \leftarrow U) \ \subseteq\ (R \cdot S) \leftarrow (V \cdot U) \tag{2.125}$$

$$(f \leftarrow g^\circ)h \ =\ f \cdot h \cdot g \tag{2.126}$$

$$k(f \leftarrow g)h \ \equiv\ k \cdot g = f \cdot h \tag{2.127}$$

In case f and g are the same,

$$f \cdot S \subseteq R \cdot f \qquad \begin{array}{ccc} A & \xleftarrow{S} & A \\ {\scriptstyle f}\downarrow & \subseteq & \downarrow{\scriptstyle f} \\ B & \xleftarrow[R]{} & B \end{array} \tag{2.128}$$

is abbreviated by $R \xleftarrow{f} S$ and we say that f has relational type $R \leftarrow S$. Pointwise: for all x and y, $x\ S\ y \Rightarrow (f\ x)\ R\ (f\ y)$.

𝔽-MONOTONICITY A *relator* $\mathbb{F}$ (Bird and de Moor, 1997) is a mathematical construction such that, for any type A, type $\mathbb{F}\ A$ is defined and for any relation $B \xleftarrow{R} A$, relation $\mathbb{F}\ B \xleftarrow{\mathbb{F}\ R} \mathbb{F}\ A$ is defined such that

$$\mathbb{F}\ id = id \tag{2.129}$$

$$\mathbb{F}\ R^\circ = (\mathbb{F}\ R)^\circ \tag{2.130}$$

$$\mathbb{F}\ (R \cdot S) = (\mathbb{F}\ R) \cdot (\mathbb{F}\ S) \tag{2.131}$$

hold.

Every time $f\ (h \leftarrow \mathbb{F}\ h)\ g$ holds — that is, $f \cdot \mathbb{F}\ h = h \cdot g$ by (2.127), h is said to be a $\mathbb{F}$-*homomorphism*.

A relation S is said to be $\mathbb{F}$-*monotone* with respect to R if relational type $R \xleftarrow{S} \mathbb{F}\ R$ holds, that is:

$$S \cdot \mathbb{F}\ R \ \subseteq\ R \cdot S \tag{2.132}$$

holds.

POWER OBJECTS Following Freyd and Scedrov (1990), a type $\mathbb{T}$ is a power object if its has a *membership* relation $A \xleftarrow{\in_{\mathbb{T}}} \mathbb{T}\ A$ such that

(a) $\in_{\mathbb{T}}$ is *straight*:

$$\frac{\in_{\mathbb{T}}}{\in_{\mathbb{T}}} = id \tag{2.133}$$

(b) $\in_{\mathbb{T}}$ is *thick*:

$$\frac{R}{\in_{\mathbb{T}}} : \mathbb{T}\ B \leftarrow A \text{ is entire for any } R : B \leftarrow A \tag{2.134}$$

Usually $\mathbb{T}$ is understood from the context and subscript $\mathbb{T}$ is omitted from $\in_{\mathbb{T}}$. Such is the case when $\mathbb{T}$ is the powerset.[1] Note that, independently of (2.134), $\frac{R}{\in} : \mathbb{T}\ B \leftarrow A$ is always simple provided (2.133) holds:

$$\begin{array}{ll} & \dfrac{R}{\in} \cdot \left(\dfrac{R}{\in}\right)^\circ \ \subseteq\ id \\ \equiv & \quad \{ \text{ symmetric division converse (2.113); power object law (2.133) } \} \\ & \dfrac{R}{\in} \cdot \dfrac{\in}{R} \ \subseteq\ \dfrac{\in}{\in} \\ \equiv & \quad \{ \text{ symmetric division (2.115) } \} \\ & true \end{array}$$

□

1 This is the case all the way through in (Bird and de Moor, 1997).

Thus (2.133,2.134) grant that

$$\Lambda R = \frac{R}{\in} \tag{2.135}$$

is a function, the pointwise

$$y = \Lambda R\ a \quad \Leftrightarrow \quad \langle \forall\ b\ ::\ b \in y\ \Leftrightarrow\ b\ R\ a \rangle$$

becomes

$$\Lambda R\ a = \{ b \mid b\ R\ a \} \tag{2.136}$$

in the case of the powerset membership.

We shall refer to (2.135) as the $\mathbb{T}$-transpose of R. Following Oliveira (2016), by (2.116) we immediately get the *fusion* law:

$$\Lambda R \cdot g = \Lambda(R \cdot g) \tag{2.137}$$

Since ΛR is a function, we reason:

$$\begin{array}{ll} & f = \Lambda R \\ \equiv & \quad \{ \text{ power-transpose definition (2.135); functional equality by (2.8) } \} \\ & f \subseteq \dfrac{R}{\in} \\ \equiv & \quad \{ \text{ symmetric division definition (2.110) } \} \\ & \in \cdot f \subseteq R \wedge f \cdot R^{\circ} \subseteq \in^{\circ} \\ \equiv & \quad \{ \text{ converses (2.28, 2.4, 2.3); shunting (2.59) } \} \\ & \in \cdot f \subseteq R \wedge R \subseteq \in \cdot f \\ \equiv & \quad \{ \text{ equality by circular inclusion (2.8) } \} \\ & \in \cdot f = R \\ \Box & \end{array}$$

So we have the universal property of *transposition*

$$f = \Lambda R \ \equiv \ \in \cdot f = R \tag{2.138}$$

and therefore cancellation,

$$\in \cdot \Lambda R = R \tag{2.139}$$

reflection,

$$\Lambda\in = id \tag{2.140}$$

which, together with a property on functions

$$\Lambda(R \cdot f) = \Lambda R \cdot f \tag{2.141}$$

gives us

$$\Lambda(\in \cdot f) = f \tag{2.142}$$

Put in another way: $(\in\cdot)$ and $(\overline{\in})$ are isomorphisms (inverse of each other) between relations of type $B \leftarrow A$ and functions of type $\mathbb{T}\ B \leftarrow A$.

Transposing a coproduct is the same as transposing its individual alternatives:

$$\Lambda[R, S] = [\Lambda R, \Lambda S] \tag{2.143}$$

The smallest relation $\bot$ of its type is such that

$$\Lambda\bot = \in \setminus \bot \tag{2.144}$$

following (2.135) and (2.114). That it is a constant function yielding the empty set follows from (2.101). At the other end of the spectrum

$$\Lambda\top = \in^\circ \,/\, \top \tag{2.145}$$

which is also a constant function yielding the largest set of all sets of outputs.

All relational symmetric divisions can be turned into the division of the two set-valued functions obtained by transposing the relations:

$$\frac{S}{R} = \frac{\Lambda R}{\Lambda S} \tag{2.146}$$

The power relator $\mathbb{P}$ is a useful tool to generalize the concept of "mapping" in functional programming languages. With relation $B \xleftarrow{R} A$, $a\ (\mathbb{P}\ R)\ b$ is true if every element of a relates to some element of b by R, and conversely. Its definition

$$\mathbb{P}\ R = \in \setminus R \cdot \in \cap \in^\circ \cdot R \,/\, \in^\circ \tag{2.147}$$

leads directly to the following rules:

$$\mathbb{P}\ R \subseteq \in \setminus R \cdot \in \tag{2.148}$$

$$\mathbb{P}\ R \subseteq \in^\circ \cdot R \,/\, \in^\circ \tag{2.149}$$

A relator distributes over composition:

$$\mathbb{P}\ (R \cdot S) = \mathbb{P}\ R \cdot \mathbb{P}\ S \tag{2.150}$$

When applied to a function h, the power relator can be expressed through the power transpose:

$$\mathbb{P}\ h = \Lambda(h \cdot \in) \tag{2.151}$$

This law is a special instance of

$$\Lambda(h \cdot S) = \mathbb{P}\ h \cdot \Lambda S \tag{2.152}$$

which leads to an interesting absorption law for coproducts and direct sums:

$$\mathbb{P}\ [f, g] \cdot \Lambda(R + S) = [\mathbb{P}\ f \cdot \Lambda R, \mathbb{P}\ g \cdot \Lambda S] \tag{2.153}$$

The free theorem of $\in$

$$\in \cdot \mathbb{P}\ R \subseteq R \cdot \in \tag{2.154}$$

can also be stated conversely:

$$\mathbb{P}\ R \cdot \in^{\circ} \subseteq \in^{\circ} \cdot R \tag{2.155}$$

(Apply the free theorem of $\in$, converses, and rename $R := R^{\circ}$.)

The union of a set of singletons is the original set

$$\mu \cdot \mathbb{P}\ \eta = id \tag{2.156}$$

from which it is easy to prove the following:

$$\mu \cdot \mathbb{P}\ (\eta \cdot R) = \mathbb{P}\ R \tag{2.157}$$

Finally, an element belongs to a union of sets if and only if it belong to one of the inner sets:

$$\in \cdot \mu = \in \cdot \in \tag{2.158}$$

GUARDS, COREFLEXIVES AND MCCARTHY'S CONDITIONAL There is a special notation for expressing a coreflexive relation that is only defined for values satisfying a predicate $Bool \xleftarrow{p} A$

$$\phi_p = id \cap \frac{true}{p} = id \cap \frac{p}{true} \tag{2.159}$$

which gives two very useful laws:

$$\phi_p \subseteq id \tag{2.160}$$

$$\phi_p \subseteq \frac{true}{p} \tag{2.161}$$

If the predicate is a negation $\neg\ p$, we can use the $false$ function instead:

$$\phi_{\neg\ p} = id \cap \frac{false}{p} = id \cap \frac{p}{false} \tag{2.162}$$

Naturally, coreflexives are idempotent:

$$\phi_p \cdot \phi_p = \phi_p \tag{2.163}$$

A common pattern involving functions can be simplified:

$$f \cdot \phi_{p \cdot f} = \phi_p \cdot f \tag{2.164}$$

Since ϕ_p is simple, statements involving relational division (universal quantification) can be transformed into statements involving relational composition (existential quantification), making sure to catch the cases where p does not hold:

$$R \ / \ \phi_p = R \cup \frac{p}{false} \ \Leftarrow \ R \text{ is connected} \tag{2.165}$$

Proof: see Appendix B.

If a relation R is entire and surjective, i.e. if R° is also entire, then it can be merged with a coreflexive.

$$R \cap \frac{p}{true} = R \cdot \phi_p \ \Leftarrow \ R \text{ is entire and surjective} \tag{2.166}$$

$$R \cap \frac{true}{p} = \phi_p \cdot R \ \Leftarrow \ R \text{ is entire and surjective} \tag{2.167}$$

For the proof of the first equality, see Appendix B. The proof for the second is almost identical, so it is omitted.

This notation for coreflexive relations is useful for representing conditional expressions. A particularly useful conditional expression is McCarthy's conditional

$$p \to R, S = [R, S] \cdot p? \tag{2.168}$$

where $p?$ is a guard — a relational product which tags a value according to condition p:

$$p? = [\phi_p, \phi_{\neg\ p}]^\circ \tag{2.169}$$

This construct can be directly translated to a guard function in Haskell:

$$p?\ x = \mathbf{if}\ p\ x\ \mathbf{then}\ i_1\ x\ \mathbf{else}\ i_2\ x$$

By biproduct absorption (2.85), we have a more friendly definition:

$$p \to R, S = R \cdot \phi_p \cup S \cdot \phi_{\neg\ p} \tag{2.170}$$

It follows by (2.160) that a McCarthy condition is smaller than the union of its branches:

$$p \to R, S \subseteq R \cup S \tag{2.171}$$

DOMAIN AND RANGE The domain and range of a relation R are the coreflexives containing all pairs (a, a) such that a is in R's domain and range, respectively. These can be defined through the $\ker$ and img operations:

$$\delta\, R = \ker R \cap id \tag{2.172}$$
$$\rho\, R = \mathrm{img}\, R \cap id \tag{2.173}$$

Each operation is associated with a Galois connection:

$$\delta\, R \subseteq S \;\equiv\; R \subseteq \top \cdot S \tag{2.174}$$
$$\rho\, R \subseteq S \;\equiv\; R \subseteq S \cdot \top \tag{2.175}$$

As expected, any relation can be composed with its domain, or compose with its range, without altering it:

$$R = R \cdot \delta\, R \tag{2.176}$$
$$R = \rho\, R \cdot R \tag{2.177}$$

Some coreflexives can be better expressed as the domain of another relation:

$$\phi_{p \cdot f} = \delta\, (\phi_p \cdot f) \tag{2.178}$$

When a function is constant, composition to the right will only limit its domain, and so:

$$f \cdot S = f \cdot \delta\, S \;\Leftarrow\; f \text{ is constant} \tag{2.179}$$

When S is entire ($\delta\, S = \top$), we have:

$$f \cdot S = f \;\Leftarrow\; f \text{ is constant and } S \text{ is entire} \tag{2.180}$$

The *true* and *false* functions are constant and have disjoint ranges, so:

$$\frac{true}{false} = \bot = \frac{false}{true} \tag{2.181}$$
$$\frac{true}{true} = \top = \frac{false}{false} \tag{2.182}$$
$$\neg\, \frac{true}{p} = \frac{false}{p} \tag{2.183}$$
$$\neg\, \frac{p}{true} = \frac{p}{false} \tag{2.184}$$

2.2 SHRINKING

One approach to solving optimization problems is to generate a set of candidate solutions and then select an optimal solution, that is, one which yields the optimal value under the appropriate objective function. To express this idea in relation algebra, Mu and Oliveira (2012) introduce the *shrinking* relational operator ($\upharpoonright$), serving a similar purpose to the $min\ R$ optimizing relation of Bird and de Moor (1997). Given relations $S : A \leftarrow B$ and $R : A \leftarrow A$, the relation $S \upharpoonright R : A \leftarrow B$, pronounced "$S$ shrunk by R", is defined as:

$$X \subseteq S \upharpoonright R \ \equiv \ \left\{ \begin{array}{l} X \subseteq S \\ X \cdot S^\circ \subseteq R \end{array} \right. \qquad \text{cf. diagram:} \qquad \begin{array}{ccc} & & B \\ & {\scriptstyle S\upharpoonright R}\swarrow & \downarrow{\scriptstyle S} \\ A & \xleftarrow[R]{} & A \end{array} \tag{2.185}$$

Here, we can think of S as relating instances of the optimization problem to the candidate solutions, we say it "produces" candidate solutions. R is a relation, typically a preorder, which compares these candidate solutions to each other. Then, the universal property of shrinking states that $S \upharpoonright R$ must only contain elements from the original relation S, and, for each of its inputs, the chosen outputs must be optimal under R when compared to other candidate solutions produced by S. In effect, the original relation is "shrunk" to include only the optimal solutions to each input.

By indirect equality, (2.185) is equivalent to the closed definition:

$$S \upharpoonright R = S \cap R \,/\, S^\circ \tag{2.186}$$

PROPERTIES OF SHRINKING Immediate from (2.185) one draws the cancellations:

$$S \upharpoonright R \subseteq S \tag{2.187}$$
$$(S \upharpoonright R) \cdot S^\circ \subseteq R \tag{2.188}$$

Cancellation (2.187) can be made into an equality adding the condition that $\mathrm{img}\ S \ \subseteq \ R$:

$$S = S \upharpoonright R \ \equiv \ \mathrm{img}\ S \subseteq R \tag{2.189}$$

If R is reflexive ($id \subseteq R$), and since all functions are simple ($\mathrm{img}\ f \subseteq id$), we have:

$$f \upharpoonright R = f \ \Leftarrow \ id \subseteq R \tag{2.190}$$

We can move functions outside a shrinking relation, or fuse them with it, like so:

$$(S \cdot f) \upharpoonright R \ = \ (S \upharpoonright R) \cdot f \tag{2.191}$$
$$(f \cdot S) \upharpoonright R \ = \ f \cdot (S \upharpoonright R_f) \tag{2.192}$$

where $R_f = f^\circ \cdot R \cdot f$. The proofs of (2.191,2.192) can be found in Appendix A of (Mu and Oliveira, 2012). Law (2.192) allows us to adjust the optimizing relation to the appropriate context. Also note that making $S, f := \in, \Lambda S$ in (2.191), applying power-transpose cancellation ($\in \cdot \Lambda S = S$), yields

$$S \upharpoonright R = (\in \upharpoonright R) \cdot \Lambda S \tag{2.193}$$

providing a short notation for a commonly used pattern. The shrunk membership relation $\in \upharpoonright R$ is what Bird and de Moor (1997) call $min\ R$ (getting the minima of a set).

Shrinking observes monotonicity on the optimizing relation (a very useful property in the sequel):

$$S \upharpoonright Q \subseteq S \upharpoonright R \ \Leftarrow \ Q \subseteq R \tag{2.194}$$

Proof:

$$\begin{array}{ll} & S \upharpoonright Q \subseteq S \upharpoonright R \\ \equiv & \quad \{ \text{ universal property (2.185) } \} \\ & \left\{ \begin{array}{l} S \upharpoonright Q \subseteq S \\ (S \upharpoonright Q) \cdot S^\circ \subseteq R \end{array} \right. \\ \equiv & \quad \{ \text{ cancellation (2.187) } \} \\ & (S \upharpoonright Q) \cdot S^\circ \subseteq R \\ \equiv & \quad \{ \text{ lower upper side (2.31) with } Q \subseteq R \ \} \\ & (S \upharpoonright Q) \cdot S^\circ \subseteq Q \\ \equiv & \quad \{ \text{ cancellation (2.188) } \} \\ & true \end{array}$$

□

On the relation itself, monotonicity is side-conditioned:

$$T \upharpoonright R \subseteq S \upharpoonright R \ \Leftarrow \ \left\{ \begin{array}{l} T \subseteq S \\ (T \upharpoonright R) \cdot S^\circ \subseteq R \end{array} \right. \tag{2.195}$$

Proof:

$$\begin{array}{ll} & T \upharpoonright R \subseteq S \upharpoonright R \\ \equiv & \quad \{ \text{ universal property (2.185) } \} \\ & \left\{ \begin{array}{l} T \upharpoonright R \subseteq S \\ (T \upharpoonright R) \cdot S^\circ \subseteq R \end{array} \right. \\ \Leftarrow & \quad \{ \text{ shrinking cancellation (2.187) } \} \end{array}$$

$$\left\{\begin{array}{l} T \subseteq S \\ (T \upharpoonright R) \cdot S^\circ \subseteq R \end{array}\right.$$

□

Moreover:

$$S \upharpoonright (Q \cap R) \subseteq (S \upharpoonright Q) \upharpoonright R \tag{2.196}$$

Proof:

$$\begin{array}{ll} & S \upharpoonright (Q \cap R) \subseteq (S \upharpoonright Q) \upharpoonright R \\ \equiv & \quad \{ \text{ universal property (2.185) } \} \\ & \left\{\begin{array}{l} S \upharpoonright (Q \cap R) \subseteq S \upharpoonright Q \\ (S \upharpoonright (Q \cap R)) \cdot (S \upharpoonright Q)^\circ \subseteq R \end{array}\right. \\ \Leftarrow & \quad \{ \text{ shrinking monotonicity with } Q \cap R \subseteq R \text{ (2.194); shrink cancellation (2.187) } \} \\ & (S \upharpoonright (Q \cap R)) \cdot S^\circ \subseteq R \\ \Leftarrow & \quad \{ \text{ shrink cancellation (2.188) } \} \\ & Q \cap R \subseteq R \\ \equiv & \quad \{ \text{ trivial } \} \\ & true \end{array}$$

□

With a specific optimizing relation and S an entire relation, shrinking can be eliminated by imposing post-condition q on S (Mu and Oliveira, 2012):

$$S \upharpoonright (\phi_q \cdot \top) = \phi_q \cdot S \;\Leftarrow\; S \text{ is entire} \tag{2.197}$$

Laws enabling shrinking of relation joins are often useful, e.g. Moreover, the "*function competition*" rule

$$(f \cup g) \upharpoonright S \;=\; (f \cap S \cdot g) \cup (g \cap S \cdot f) \tag{2.198}$$

tells how two functions compete with each other to produce optimized results. (Recall that $S / g^\circ = S \cdot g$.) Other laws enabling shrinking of relation joins are often useful, e.g.

$$(P \cup S) \upharpoonright R = (P \upharpoonright R) \cup (S \upharpoonright R) \;\Leftarrow\; P \cdot S^\circ \subseteq \bot \tag{2.199}$$

which is a corollary of

$$(R \cup S) \upharpoonright Q = (R \upharpoonright Q) \cap Q / S^\circ \cup (S \upharpoonright Q) \cap Q / R^\circ \tag{2.200}$$

Proof: see (Oliveira and Ferreira, 2013).

Shrinking distributes through other relational constructs in the way one would expect:

$$[S, T] \upharpoonright R \;=\; [S \upharpoonright R, T \upharpoonright R] \tag{2.201}$$

Proof: see (Mu and Oliveira, 2012).
For shrinking to distribute through relational direct sums, the optimization criterion must operate independently on the resulting types, i.e. it must be a direct sum as well:

$$(S + T) \upharpoonright (R + Q) = S \upharpoonright R + T \upharpoonright Q \tag{2.202}$$

Proof: see Appendix B.

Generalizing from exercise 7.15 from Bird and de Moor (1997) on pairing shrinking:

$$\langle S \upharpoonright R, T \upharpoonright Q \rangle \subseteq \langle S, T \rangle \upharpoonright (R \times Q) \tag{2.203}$$

Proof: see Appendix B.

2.3 RECURSIVE RELATIONS

This section introduces two general recursion patterns and their associated laws and properties. Along the way, a few examples of how common functions can be expressed using them are given.

2.3.1 *Catamorphisms and anamorphisms*

A *catamorphism* is a general construct allowing the transformation of a recursive datatype into any other type, usually in a destructive process, as the word's origin κατά (Greek for "downwards") indicates. An *anamorphism* is the converse of a catamorphism [2] , which means it is able to generate a complex structure from a simpler one.

A 'recursive' datatype, in this sense, is a datatype $\mathbb{T}$ for which there exists an *initial* $\mathbb{F}$-algebra $\mathbb{T} \xleftarrow{\mathsf{in}} \mathbb{F}\ \mathbb{T}$. As the two are so intimately related, we can write type $\mathbb{T}$ as $\mu_{\mathbb{F}}$. An $\mathbb{F}$-algebra is any relation of type $A \leftarrow \mathbb{F}\ A$, and an initial $\mathbb{F}$-algebra for (initial) type $\mathbb{T}$, which can thought of as its constructor, is when there exists a unique relation of type $A \leftarrow \mathbb{T}$, written $(\!|\, R \,|\!)$, for any other $\mathbb{F}$-algebra $A \xleftarrow{R} \mathbb{F}\ A$, the "gene" of the catamorphism, such that

$$(\!|\, R \,|\!) \cdot \mathsf{in} = R \cdot \mathbb{F}\ (\!|\, R \,|\!) \tag{2.204}$$

2 In category theoretic terms, an anamorphism is the unique homomorphism from a coalgebra to the final coalgebra of a functor, while a catamorphism is the unique homomorphism from its initial algebra to another algebra. So, using the term 'anamorphism' for the converse of a catamorphism is only acceptable when initial algebras and final coalgebras coincide, ie. have the same carrier. By Theorem 3.8 of (Hasuo et al., 2007), the initial F-algebra in Sets yields the final $\mathbb{F}$-coalgebra in Rel, under rather mild conditions on functor $\mathbb{F}$. Therefore, it makes sense in Rel to use the term 'anamorphism' as a synonym of converse of a catamorphism.

holds. Catamorphisms, also called folds in some of the literature, exhibit the following *universal property*,

$$X = (\!|R|\!) \;\equiv\; X \cdot \mathsf{in} = R \cdot (\mathbb{F}\ X) \qquad (2.205)$$

$$\begin{array}{ccc} \mathbb{T} & \xleftarrow{\mathsf{in}} & \mathbb{F}\ \mathbb{T} \\ {\scriptstyle (\!|R|\!)}\downarrow & & \downarrow{\scriptstyle \mathbb{F}\ (\!|R|\!)} \\ B & \xleftarrow[R]{} & \mathbb{F}\ B \end{array}$$

The base relator $\mathbb{F}$ of type $\mathbb{T}$ captures its recursive pattern. For instance, for finite lists which hold elements of type A, one has

$$\begin{cases} \mathbb{F}\ X = 1 + A \times X \\ \mathbb{F}\ f = id + id \times f \end{cases} \qquad (2.206)$$

$$\mathsf{in} = [nil, cons] \qquad (2.207)$$

where

$$nil\ \_ = [\,]$$
$$cons\ (h, t) = h : t$$

and nil and $cons$ have disjoin ranges:

$$cons^\circ \cdot nil = \bot \qquad (2.208)$$

The following are useful examples of relational catamorphisms over lists: the list-prefix relation

$$\preceq : A^* \leftarrow A^* \qquad (2.209)$$
$$\preceq = (\!|\, [nil, cons \cup nil] \,|\!) \qquad (2.210)$$

the subsequence relation

$$(\sqsubseteq) : A^* \leftarrow A^*$$
$$(\sqsubseteq) = (\!|\, [nil, cons \cup \pi_2] \,|\!) \qquad (2.211)$$

and, for non-empty lists, where

$$\begin{cases} \mathbb{F}\ X = A + A \times X \\ \mathbb{F}\ f = id + id \times f \end{cases}$$
$$\mathsf{in} = [wrap, cons]$$

the membership relation

$$\epsilon_+ : A \leftarrow A^+$$
$$\epsilon_+ = (\!|\, [id, \pi_1 \cup \pi_2] \,|\!) \qquad (2.212)$$

Two properties stem from (2.205) that prove particularly useful in calculations about $(\!|\, R \,|\!)$, namely *fusion*

$$S \cdot (\!|\, R \,|\!) = (\!|\, Q \,|\!) \;\Leftarrow\; S \cdot R = Q \cdot \mathbb{F}\, S \tag{2.213}$$

and *cancellation*, already given above (2.204). Fusion provides a sufficient condition on S, R and Q for merging $S \cdot (\!|\, R \,|\!)$ into $(\!|\, Q \,|\!)$. This is incredibly useful for program derivation, because any catamorphism can be implemented as a computer program, a consequence of a result known as the Eilenberg-Wright Lemma, which we introduce at the end of this section.

By indirect equality one derives, from cata-cancellation:

$$X \subseteq (\!|\, R \,|\!) \;\Leftarrow\; X \cdot \mathsf{in} \subseteq R \cdot \mathbb{F}\, X \tag{2.214}$$

$$(\!|\, R \,|\!) \subseteq X \;\Leftarrow\; R \cdot \mathbb{F}\, X \subseteq X \cdot \mathsf{in} \tag{2.215}$$

These lead to “weaker" versions of cata-fusion:

$$Q \cdot (\!|\, S \,|\!) \subseteq (\!|\, R \,|\!) \;\Leftarrow\; Q \cdot S \subseteq R \cdot \mathbb{F}\, Q \tag{2.216}$$

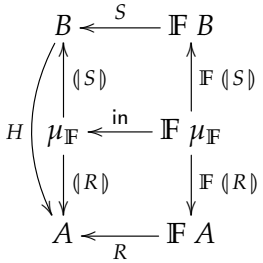
$$(\!|\, R \,|\!) \subseteq Q \cdot (\!|\, S \,|\!) \;\Leftarrow\; R \cdot \mathbb{F}\, Q \subseteq Q \cdot S \tag{2.217}$$

2.3.2 *Hylomorphisms*

A *hylomorphism* is a more expressive recursive pattern than a catamorphism, because it is the composition of a catamorphism with an anamorphism,

$$H = (\!|\, R \,|\!) \cdot (\!|\, S \,|\!)^{\circ} \tag{2.218}$$

which can be thought of as phases of the hylomorphism. With $\mathbb{F}$-algebras $A \xleftarrow{R} \mathbb{F}\, A$ and $B \xleftarrow{S} \mathbb{F}\, B$, $A \xleftarrow{H} B$ can be diagrammed as:



The intermediate type $\mu_{\mathbb{F}}$ produced by the anamorphism is known as the *virtual data structure* of the hylomorphism. H is the least fixpoint of the relational equation

$$X = R \cdot \mathbb{F}\, X \cdot S^{\circ} \tag{2.219}$$

and, as such, any hylomorphism on either side of a relational inequality can be substituted by its least fixpoint, leading to

$$(\!|R|\!) \cdot (\!|S|\!)^\circ \subseteq X \;\Leftarrow\; R \cdot \mathbb{F}\, X \cdot S^\circ \subseteq X \tag{2.220}$$

$$X \subseteq (\!|R|\!) \cdot (\!|S|\!)^\circ \;\Leftarrow\; X \subseteq R \cdot \mathbb{F}\, X \cdot S^\circ \tag{2.221}$$

in effect requiring proof that the statement holds through one recursive step of the hylomorphism, which includes both phases.

While the membership of non-empty finite lists, $A \xleftarrow{\epsilon_+} A^+$, is, as we have seen, a catamorphism, the same is not true for (possibly empty) finite lists, $A \xleftarrow{\epsilon} A^*$. The enticing definition $\epsilon = (\!|[\bot, \pi_1 \cup \pi_2]|\!)$ is incorrect because the base case (the empty list) is not defined, and so the relation "bottoms-out":

$$\begin{array}{ll}
 & (\!|[\bot, \pi_1 \cup \pi_2]|\!) = \bot \\
\equiv & \quad \{ \text{ circular equality (2.8); } \bot \text{ below everything (2.51) } \} \\
 & (\!|[\bot, \pi_1 \cup \pi_2]|\!) \subseteq \bot \\
\Leftarrow & \quad \{ \text{ cata fusion (2.215) } \} \\
 & [\bot, \pi_1 \cup \pi_2] \cdot \mathbb{F}\, \bot \subseteq \bot \cdot \mathsf{in} \\
\equiv & \quad \{ \text{ coproducts (2.90) } \} \\
 & \left\{ \begin{array}{l} \bot \subseteq \bot \cdot nil \\ (\pi_1 \cup \pi_2) \cdot (id \times \bot) \subseteq \bot \cdot cons \end{array} \right. \\
\equiv & \quad \{ \text{ } \bot \text{ is zero of composition (2.57) (twice) } \} \\
 & (\pi_1 \cup \pi_2) \cdot (id \times \bot) \subseteq \bot \\
\equiv & \quad \{ \text{ } (id \times \bot) = \pi_1^\circ \cdot \pi_1 \cap \pi_2^\circ \cdot \bot \cdot \pi_2 = \pi_1^\circ \cdot \pi_1 \cap \bot = \bot \text{ } \} \\
 & (\pi_1 \cup \pi_2) \cdot \bot \subseteq \bot \\
\equiv & \quad \{ \text{ } \bot \text{ is zero of composition (2.57) } \} \\
 & \bot \subseteq \bot \\
\Box &
\end{array}$$

Instead, a definition is given by stating that either an element of a list is the head of the list, or it is contained in its tail, expressed relationally by

$$\epsilon = Hd \cup \epsilon \cdot Tl \tag{2.222}$$

where $Hd = \pi_1 \cdot cons^\circ$ and $Tl = \pi_2 \cdot cons^\circ$. This is an example of a tail-recursive relational hylomorphism, where $\mathbb{F}\, X = A + X$:

$$\epsilon = (\!|[id, id]|\!) \cdot (\!|[Hd^\circ, Tl^\circ]|\!)^\circ \tag{2.223}$$

The following equation helps simplify proofs involving finite list membership:

$$\epsilon \cdot \mathsf{in} = [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \tag{2.224}$$

Proof:

$$\begin{array}{ll}
& \epsilon \cdot \mathsf{in} = [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \\
\equiv & \quad \{ \text{ def-}\epsilon \text{ (2.222)} ; \mathsf{in} = [nil, cons] \text{ (2.207)} \ \} \\
& (Hd \cup \epsilon \cdot Tl) \cdot [nil, cons] = [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \\
\equiv & \quad \{ \text{ join bilinearity (2.21); coproduct equality (2.80) } \} \\
& \left\{ \begin{array}{l} (Hd \cup \epsilon \cdot Tl) \cdot nil = \bot \\ (Hd \cup \epsilon \cdot Tl) \cdot cons = \pi_1 \cup \epsilon \cdot \pi_2 \end{array} \right. \\
\equiv & \quad \{ \ Hd = \pi_1 \cdot cons^\circ, Tl = \pi_2 \cdot cons^\circ \text{; join bilinearity (2.21) } \} \\
& \left\{ \begin{array}{l} (\pi_1 \cup \epsilon \cdot \pi_2) \cdot cons^\circ \cdot nil = \bot \\ (\pi_1 \cup \epsilon \cdot \pi_2) \cdot cons^\circ \cdot cons = \pi_1 \cup \epsilon \cdot \pi_2 \end{array} \right. \\
\equiv & \quad \{ \text{ (2.208); } cons \text{ is injective, } \ker cons = id \ \} \\
& \left\{ \begin{array}{l} \bot = \bot \\ \pi_1 \cup \epsilon \cdot \pi_2 = \pi_1 \cup \epsilon \cdot \pi_2 \end{array} \right. \\
\Box &
\end{array}$$

2.4 IMPLEMENTING RELATION CALCULUS WITH FUNCTIONS

So far, some constructs which use relations have been introduced with the aim of specifying DP problems and allowing us to easily deal with non-determinism algebraically. The typical method for most programmers in implementing DP algorithms using non-determinism is to explicitly manipulate a collection of solutions between iterations of the algorithm. However, this can be done implicitly using monads, specifically the powerset monad. In this section, rules for transforming these relational constructs into functional ones using the powerset monad are introduced.

THE POWERSET MONAD A monad is defined by three things: a datatype $\mathbb{T}$, a unit function $\mathbb{T}\ A \xleftarrow{\eta} A$, and a multiplication function $\mathbb{T}\ A \xleftarrow{\mu} \mathbb{T}^2\ A$. In the case of the powerset monad, the datatype $\mathbb{T}$ is $\mathbb{P}$ (powerset) and the unit and multiplication functions are the singleton and the union, respectively:

$$\eta_{\mathbb{P}}\ a = \{a\} \tag{2.225}$$

$$\mu_{\mathbb{P}} = \bigcup \tag{2.226}$$

The equivalent of regular relation composition, monadic composition $\mathbb{T}\ C \xleftarrow{f \bullet g} A$, with monadic arrows $\mathbb{T}\ B \xleftarrow{g} A$ and $\mathbb{T}\ C \xleftarrow{f} B$, is defined as:

$$f \bullet g = \mu \cdot \mathbb{T}\, f \cdot g \tag{2.227}$$

In Haskell, monadic composition can be encoded using the bind operator ($\gg=$) or the syntactically friendlier **do** notation facilitating the "chaining" of operations in an imperative style:

$$(f \bullet g)\ x = g\ x \gg= f = \mathbf{do}\ \{a \leftarrow g\ x; f\ a\} \tag{2.228}$$

In the powerset monad, monadic composition is the composition of two set-valued functions. Each element in the set produced by g is turned into a new set by the application of f, resulting in a set of sets which is then unified into a single set containing all elements. To make use of this, we must turn our relations into set-valued functions. This is done using the power transpose operator:

$$\Lambda R\ a = \{b \mid b\ R\ a\} \tag{2.229}$$

Any set type forms a monoid, i.e. there is an associative binary operation $plus$ with an identity element $zero$. In Haskell, this is implemented via the $MonadPlus$ type class, with the $mplus$ ($\oplus$) operator and the $mzero$ constant function. For the powerset, they are defined as the set union binary operator and its identity element, the empty set:

$$x \oplus y = x\ \cup_{\mathbb{P}}\ y \tag{2.230}$$

$$mzero = \{\} \tag{2.231}$$

A few rules involving common operations will be used in the final part of the process of deriving algorithmic solutions from specifications. In addition to general rules involving relatively simple constructs introduced earlier, it is also important to be able to transpose other types of relations, including more complex ones such as relational union, intersection and division:

$$\Lambda(R \cdot S) = \Lambda R \bullet \Lambda S \tag{2.232}$$

$$\Lambda id = \eta \tag{2.233}$$

$$\Lambda f = \eta \cdot f \tag{2.234}$$

$$\Lambda \bot = \underline{mzero} \tag{2.235}$$

$$\Lambda(S \cup R)\ a = (\Lambda S\ a) \oplus (\Lambda R\ a) \tag{2.236}$$

$$\Lambda(f \cup g)\ a = \{f\ a, g\ a\} \tag{2.237}$$

$$\Lambda(R \cap S)\ a = \{b \mid b \in \Lambda R\ a \wedge b \in \Lambda S\ a\} \tag{2.238}$$

$$\Lambda(R\ /\ S)\ a = \{b \mid \langle \forall\ c\ :\ c \in \Lambda S^{\circ}\ a :\ b \in \Lambda R\ c \rangle\} \tag{2.239}$$

$$\Lambda(R \times S)\ (a,b) = \{(c,d) \mid c \in \Lambda R\ a\ \wedge\ d \in \Lambda R\ b\} \tag{2.240}$$

Upon application of these rules, $b \in \Lambda R\ a$ may be implicitly converted to the equivalent expression $b\ R\ a$ if R doesn't need to be transposed in the given context.

The last rule is often used with $R := id$, which yields $c = a$ on the right side. This can be simplified through the use of a one-point rule: given a relational expression $E(x)$ and a relational predicate $P(x)$, we have:

$$\{E\ (a) \mid a = b\ \wedge\ P\ (a)\}\ \equiv\ \{E\ (b) \mid P\ (b)\} \tag{2.241}$$

As a first example of transposition we calculate the popular function $elems : \mathbb{P}\ A \leftarrow A^*$ that yields the set of all elements of a finite list, which arises as

$$elems = \Lambda\epsilon \tag{2.242}$$

By transposing both sides of (2.224) we obtain the usual, inductive definition:

$$\begin{array}{ll}
 & \Lambda(\epsilon \cdot \mathsf{in}) = \Lambda[\bot, \pi_1 \cup \epsilon \cdot \pi_2] \\
\equiv & \quad \{\ (2.141);\ elems = \Lambda\epsilon;\ \text{coproduct power-transpose (2.143)}\ \} \\
 & elems \cdot \mathsf{in} = [\Lambda\bot, \Lambda(\pi_1 \cup \epsilon \cdot \pi_2)] \\
\equiv & \quad \{\ \text{coproducts; go pointwise; } \Lambda\bot\ _ = mzero = \{\}\ \} \\
 & \left\{ \begin{array}{l} elems\ [\,] = \{\} \\ elems\ (h : t) = \Lambda(\pi_1 \cup \epsilon \cdot \pi_2)\ (h, t) \end{array} \right. \\
\equiv & \quad \{\ (2.236);\ (2.141);\ elems = \Lambda\epsilon;\ \Lambda f = \eta \cdot f\ \} \\
 & \left\{ \begin{array}{l} elems\ [\,] = \{\} \\ elems\ (h : t) = \{h\}\ \cup_{\mathbb{P}}\ elems\ t \end{array} \right.
\end{array}$$

A coreflexive relation can be naturally represented by a set-valued function — a singleton if the condition holds, or the empty set if it doesn't. This is commonly known as the $guard$ function.

$$\Lambda\phi_p = guard\ p = p \rightarrow \eta, \underline{mzero} \tag{2.243}$$

Proof:

$$\begin{array}{ll}
 & \Lambda\phi_p \\
= & \quad \{\ \phi_p = i_1^\circ \cdot p?\ \text{since}\ p? = [\phi_p, \phi_{\neg\, p}]^\circ\ (2.169)\ \} \\
 & \Lambda(i_1^\circ \cdot p?) \\
= & \quad \{\ i_1^\circ = [id, \bot]\ (2.76)\ \} \\
 & \Lambda([id, \bot] \cdot p?) \\
= & \quad \{\ \Lambda(R \cdot f) = \Lambda R \cdot f\ (2.137)\ \} \\
 & \Lambda[id, \bot] \cdot p? \\
= & \quad \{\ \text{power-transpose of coproduct (2.143)}\ \}
\end{array}$$

$$
\begin{array}{ll}
& [\Lambda id\,,\Lambda\bot]\cdot p? \\
= & \quad \{\ \Lambda id = \eta \text{ (2.233)}; \Lambda\bot = \underline{mzero} \text{ (2.235)}\ \} \\
& [\eta\,,\underline{mzero}]\cdot p? \\
= & \quad \{\ \text{McCarthy conditional (2.168)}\ \} \\
& p \to \eta, \underline{mzero} \\
\Box &
\end{array}
$$

THE EILENBERG-WRIGHT LEMMA A set-valued catamorphism must have a gene that forms a set from recursively calculated sets. The Eilenberg-Wright Lemma establishes a relation between relational and functional catamorphisms by doing precisely this:

$$X = (\!|\, R \,|\!) \ \equiv\ \Lambda X = (\!|\, \Lambda(R\cdot \mathbb{F}\in) \,|\!) \tag{2.244}$$

This arises as corollary of the adjoint-fold theorem of (Oliveira, 2019) instantiated to the adjunction underlying the powerset transpose construction (2.138).

This lemma is useful wherever refinement from relational catamorphisms does not shrink to a function, making it necessary to encode the resulting catamorphism in terms of a set-valued function. The particular case of relational catamorphisms over lists pops up so often that it is useful to expand the Eilenberg-Wright encoding for this data type:

$$
\begin{array}{ll}
& X = (\!|\, R \,|\!) \\
\equiv & \quad \{\ \text{Eilenberg-Wright Lemma (2.244)}\ \} \\
& \Lambda X = (\!|\, \Lambda(R\cdot \mathbb{F}\in) \,|\!) \\
\equiv & \quad \{\ \mathbb{F}\ R = id + id\times R \text{ assumed, then } R := [R_1\,,R_2]\ \} \\
& \Lambda X = (\!|\, \Lambda[R_1\,,R_2\cdot(id\times\in)] \,|\!) \\
\equiv & \quad \{\ \text{coproduct power-transpose (2.143)}\ \} \\
& \Lambda X = (\!|\, [\Lambda R_1\,,\Lambda(R_2\cdot(id\times\in))] \,|\!) \\
\equiv & \quad \{\ \text{universal property (2.205)}\ \} \\
& \left\{ \begin{array}{l} \Lambda X\cdot nil = \Lambda R_1 \\ \Lambda X\cdot cons = \Lambda(R_2\cdot(id\times\in))\cdot(id\times\Lambda X) \end{array} \right. \\
\equiv & \quad \{\ \text{power-transpose of composition (2.232)}\ \} \\
& \left\{ \begin{array}{l} \Lambda X\cdot nil = \Lambda R_1 \\ \Lambda X\cdot cons = \Lambda R_2 \bullet \Lambda(id\times\in)\cdot(id\times\Lambda X) \end{array} \right. \\
\equiv & \quad \{\ \text{go pointwise}\ \}
\end{array}
$$

$$
\begin{cases}
\Lambda X\ [\,] = \Lambda R_1\ \_ \\
\Lambda X\ (h:t) = (\Lambda R_2 \bullet \Lambda(id \times \in))\ (h, \Lambda X\ t)
\end{cases}
$$

$$
\equiv \quad \{\ \text{introduce } \mathbf{do} \text{ notation (2.228) and simplify}\ \}
$$

$$
\begin{cases}
\Lambda X\ [\,] = \Lambda R_1\ () \\
\Lambda X\ (h:t) = \mathbf{do}\ \{b \leftarrow \Lambda X\ t; \Lambda R_2\ (h, b)\}
\end{cases}
$$

Summing up:

$$
X = (\!|\,[R_1, R_2]\,|\!) \ \equiv\ \begin{cases}
\Lambda X\ [\,] = \Lambda R_1\ () \\
\Lambda X\ (h:t) = \mathbf{do}\ \{b \leftarrow \Lambda X\ t; \Lambda R_2\ (h, b)\}
\end{cases} \tag{2.245}
$$

2.5 METAPHORS AND METAPHORISMS

Substituting $R := id$ in (2.5) and renaming, we get a *metaphor* $g^\circ \cdot f$,

$$
b\ (g^\circ \cdot f)\ a \quad \Leftrightarrow \quad g\ b = f\ a \tag{2.246}
$$

conveying the preservation of some property, observed by f on the input and by g on the output. A metaphor can be expressed as a symmetric division (Oliveira, 2018), so we have the following notation:

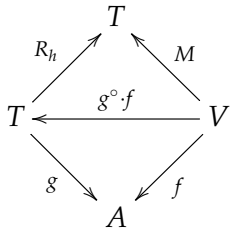
$$
\frac{f}{g} \ = \ g^\circ \cdot f \tag{2.247}
$$

Since a metaphor is a symmetric division, previously introduced properties also hold in metaphors. Applying properties (2.116) and (2.117) to functions results in the following property on metaphors:

$$
\frac{id}{g} \cdot \frac{h}{k} \cdot \frac{f}{id} = \frac{h \cdot f}{k \cdot g} \tag{2.248}
$$

A shrinking metaphor is a useful pattern for specifying some optimization problems:

$$
M = \frac{f}{g} \upharpoonright R_h \tag{2.249}
$$



The inner metaphor represents the algorithm's invariant, while the optimizing relation R_h indicates the best solution, ordering elements through a measure given by function h. For example, Oliveira (2018) specifies the problem of building trees of minimum height as $\frac{tips}{tips} \upharpoonright (\leqslant)_{height}$, where $tips$ is a function that builds the sequence of a tree's leaves.

If, instead of a function, a relation is used in either side of the metaphor, it will be shrunk to its deterministic fragment $R \upharpoonright id$ (Mu and Oliveira, 2012):

$$\frac{R}{g} = g^\circ \cdot (R \upharpoonright id) \tag{2.250}$$

$$\frac{f}{R} = (R \upharpoonright id)^\circ \cdot f \tag{2.251}$$

POST-CONDITIONED METAPHORS Pattern of metaphor shrinking from (Oliveira, 2018)

$$\frac{f}{g} \upharpoonright \frac{true}{q} \tag{2.252}$$

indicating that only the outputs satisfying q are regarded as good enough. Oliveira (2018) shows that (2.252) is equivalent to

$$\phi_q \cdot \frac{f}{g} \tag{2.253}$$

a pattern referred to as a *postconditioned metaphor*.

METAPHORISMS Metaphorisms are simply metaphors in which f and g (here renamed) are catamorphisms:

$$\frac{(\!|f|\!)}{(\!|g|\!)} = (\!|g|\!)^\circ \cdot (\!|f|\!) \tag{2.254}$$

Note the difference to hylomorphisms. Here, reading composition as existential quantification (2.1) is useful: for every input-output pair, there must exist a value that is both observed by $(\!|f|\!)$ on the input and observed by $(\!|g|\!)$ on the output. This is the invariant property specified by the metaphorism.

As with metaphors, there is a similar *shrinking metaphorism* pattern, which now divides the problem into an invariant on inductive types and an optimization phase. The specification for the minimum height tree problem given before, $\frac{tips}{tips} \upharpoonright (\leqslant)_{height}$, is actually an example of a shrinking metaphorism, as $tips$ can be defined as a catamorphism.

Metaphorisms of the form $\frac{f}{f}$, where $f = (\!|g|\!)$, are $\mathbb{F}$-congruences on the initial algebra of f:

$$\mathsf{in} \cdot \mathbb{F}\,\frac{f}{f} \subseteq \frac{f}{f} \cdot \mathsf{in} \tag{2.255}$$

Proof:

$$\begin{aligned} & \mathsf{in} \cdot \mathbb{F}\,\frac{f}{f} \subseteq \frac{f}{f} \cdot \mathsf{in} \\ \equiv & \quad \{\ (2.112);\ \text{shunting } (2.58);\ \text{catamorphism } (2.204)\ \text{(twice)}\ \} \\ & g \cdot \mathbb{F}\,f \cdot \mathbb{F}\,(f^\circ \cdot f) \subseteq g \cdot \mathbb{F}\,f \end{aligned}$$

$$\equiv \quad \{ \text{ functors; } f \text{ is simple, so } \operatorname{img} f = id \text{ (2.41) } \}$$
$$g \cdot \mathbb{F}\, f \subseteq g \cdot \mathbb{F}\, f$$
$$\square$$

2.6 ALGEBRA OF PROGRAMMING

2.6.1 *The converse of a function theorem*

Some problems are easily specified by the converse of a function. For instance, the square root is the converse of the square function $sq\ x = x^2$, i.e. $Sqrt = sq^\circ$. Provided some conditions hold, one can transform function converses into relational catamorphisms.

Theorem 2.1. *Let* $T \xleftarrow{\mathsf{in}} \mathbb{F}\, T$ *and* $T \xleftarrow{f} A$ *be given. Then, with* $R : A \leftarrow \mathbb{F}\, A$ *surjective:*

$$f^\circ = (\!|\, R \,|\!) \ \Leftarrow \ f \cdot R \subseteq \mathsf{in} \cdot \mathbb{F}\, f \tag{2.256}$$

Proof: see theorem 6.4 in (Bird and de Moor, 1997). $\square$

2.6.2 *The Greedy Theorem*

The Greedy Theorem can be applied to problems where a greedy strategy arrives at the optimal solution, that is, where choosing the best solution out of all possible candidate solutions can be done by maintaining, at each step of the algorithm, a single candidate solution, specifically, the one that results from a *locally optimal* decision.

Theorem 2.2. *Greedy theorem:*

$$(\!|\, S \upharpoonright R \,|\!) \ \subseteq \ (\!|\, S \,|\!) \upharpoonright R \tag{2.257}$$

holds provided R *is transitive and* S *is monotonic with respect to* R° *(2.132).*
Proof: see theorem 7.2 in (Bird and de Moor, 1997). $\square$

2.6.3 *Dynamic Programming Theorems*

It is almost always advantageous to reduce complexity by reasoning with simpler constructs. A shrinking hylomorphism can be refined to a least fixed point not involving the catamorphism or anamorphism, provided no candidate solutions are excluded from consideration in the resulting equation. Additionally, monotonicity with respect to the optimizing relation must hold.

Theorem 2.3. *With $H = (\!|\, S \,|\!) \cdot (\!|\, T \,|\!)^\circ$ over functor $\mathbb{F}$, we have:*

$$\langle \mu\ X\ ::\ S \cdot \mathbb{F}\ X \cdot T^\circ \upharpoonright R \rangle\ \subseteq\ H \upharpoonright R \tag{2.258}$$

provided S is monotonic with respect to R, and $\delta\ T \subseteq \delta\ (S \cdot \mathbb{F}\ M)$.
Proof: see theorem 2 in (Mu and Oliveira, 2012) □

(Reminder: the expression $S \cdot \mathbb{F}\ X \cdot T^\circ \upharpoonright R$ is same as $(S \cdot \mathbb{F}\ X \cdot T^\circ) \upharpoonright R$ due to composition binding tighter than any other binary combinator.)

When S is a function h monotonic with respect to R, the domain restriction is automatically satisfied.

Theorem 2.4. *Given $H = (\!|\, h \,|\!) \cdot (\!|\, T \,|\!)^\circ$ and $R \xleftarrow{h} \mathbb{F}\ R$. Then*

$$\langle \mu\ X\ ::\ h \cdot \mathbb{F}\ X \cdot T^\circ \upharpoonright R \rangle\ \subseteq\ H \upharpoonright R \tag{2.259}$$

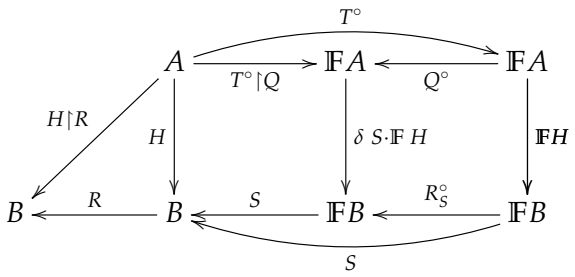
Proof: see theorem 9.1 in (Bird and de Moor, 1997)

An optimal solution can be extracted from an intermediate structure if we find an optimization relation for the intermediate structure that is consistent: an optimal structure under this relation will result in an optimal solution under the original optimization relation.

Theorem 2.5. *Let $H = (\!|\, S \,|\!) \cdot (\!|\, T \,|\!)^\circ$ where S is a simple relation We have*

$$\langle \mu\ X\ ::\ S \cdot \mathbb{F}X \cdot (T^\circ \upharpoonright Q) \rangle\ \subseteq\ H \upharpoonright R$$

provided S is monotonic on R and $S \cdot \mathbb{F}H \cdot Q^\circ \subseteq R^\circ \cdot S \cdot \mathbb{F}H$, cf. the following type diagram where R°_S abbreviates $S^\circ \cdot R^\circ \cdot S$:



The diagram assumes the following rule, valid for S simple:

$$S \cdot R \subseteq T\ \ \equiv\ \ \delta\ S \cdot R \subseteq S^\circ \cdot T \tag{2.260}$$

Proof: *see (Mu and Oliveira, 2012).*[3]
□

Finally, the following theorem gives a condition for proving monotonicity under a relation expressed as fold, which is helpful in applying the other theorems.

3 This is a slight generalization of theorem 10.1 of (Bird and de Moor, 1997).

Theorem 2.6. *Let* $R = (\!|\, I \,|\!)$ *where* $\mathsf{in} \subseteq I$. *Then* R *is monotone with respect to* in*:*

$$\mathsf{in} \cdot \mathbb{F}\, R \subseteq R \cdot \mathsf{in} \;\Leftarrow\; \mathsf{in} \subseteq I \tag{2.261}$$

Proof: see (Mu and Oliveira, 2012). □

2.7 TOWARDS EXECUTABILITY

One consequence of using shrinking to specify and refine relational inductive relations is that non-determinism is reduced by the implicit optimization and one is lead to runnable inductive relations, e.g. *functional* catamorphisms. It often happens, though, that the process does not end in a function, meaning that the only way to implement the outcome is to transpose it, recall (2.193) and (2.141),

$$\Lambda(S \upharpoonright R) = \Lambda(\in \upharpoonright R) \cdot \Lambda S \tag{2.262}$$

that is:

$$\mathbb{P}\, B \xleftarrow{\Lambda(\in \upharpoonright R)} \mathbb{P}\, B \xleftarrow{\Lambda S} A$$

For finite B, power-type $\mathbb{P}\, B$ can be implemented in several ways, e.g. by B^*, in which case $\epsilon = Hd \cup \epsilon \cdot Tl$ — recall (2.222).

Let us see how to implement $\Lambda(\epsilon \upharpoonright R)$, using theorem 2.4. By (2.222), $\mathbb{F}\, R = id + R$ and $h = [id, id]$ and pre-condition $R \xleftarrow{h} \mathbb{F}\, R$ clearly holds provided R is reflexive. By (2.259):

$$\begin{array}{ll}
 & \langle \mu\, X :: ([id,id] \cdot (id + X) \cdot [Hd^\circ, Tl^\circ]^\circ) \upharpoonright R \rangle \subseteq \epsilon \upharpoonright R \\
\equiv & \quad \{ \text{ definition of } Hd \text{ and } Tl \text{ (3.7, 3.8); coproduct fusion (2.83); converse } \} \\
 & \langle \mu\, X :: ([id,id] \cdot (id + X) \cdot [\pi_1^\circ, \pi_2^\circ]^\circ \cdot cons^\circ) \upharpoonright R \rangle \subseteq \epsilon \upharpoonright R \\
\Leftarrow & \quad \{ \text{ shrink monotonicity (2.195); direct sum and biproduct absorption (2.87, 2.85) } \} \\
 & \langle \mu\, X :: [\bot, \pi_1 \cup X \cdot \pi_2] \cdot [nil, cons]^\circ \upharpoonright R \rangle \subseteq \epsilon \upharpoonright R \\
\equiv & \quad \{ \text{ function shrinking (2.191) } \} \\
 & \langle \mu\, X :: ([\bot, \pi_1 \cup X \cdot \pi_2] \upharpoonright R) \cdot \mathsf{out} \rangle \subseteq \epsilon \upharpoonright R \\
\equiv & \quad \{ \text{ coproduct shrinking (2.201) } \} \\
 & \langle \mu\, X :: [\bot, \underbrace{(\pi_1 \cup X \cdot \pi_2) \upharpoonright R}_{S}] \cdot \mathsf{out} \rangle \subseteq \epsilon \upharpoonright R
\end{array}$$

Thus we reach the inductive relation X such that

$$X \cdot \mathsf{in} = [\bot, S]$$

As is evident below, S can be transformed in a way that will enable the definition of the power transpose of X as a catamorphism:

$$
\begin{array}{ll}
& (\pi_1 \cup X \cdot \pi_2) \upharpoonright R \\
= & \quad \{\ X = \in \cdot \Lambda X \text{ (2.139)}\ \} \\
& (\pi_1 \cup \in \cdot \Lambda X \cdot \pi_2) \upharpoonright R \\
= & \quad \{\ \text{free theorems of } \pi_1 \text{ and } \pi_2 \text{ (2.74, 2.75)}\ \} \\
& (\pi_1 \cdot (id \times \Lambda X) \cup \in \cdot \pi_2 \cdot (id \times \Lambda X)) \upharpoonright R \\
= & \quad \{\ \text{function shrinking (2.191)}\ \} \\
& \underbrace{(\pi_1 \cup \in \cdot \pi_2) \upharpoonright R}_{T} \cdot (id \times \Lambda X)
\end{array}
$$

where T unfolds to

$$T = \pi_1 \cap R \,/\, (\in \cdot \pi_2)^\circ \cup \in \cdot \pi_2 \cap R \cdot \pi_1$$

by (2.200).

We finish off by taking taking the transpose of both sides of the equation giving us a functional implementation of the least prefix point of $\epsilon \upharpoonright R$:

$$
\begin{array}{ll}
& X \cdot \mathsf{in} = [\bot, S] \\
= & \quad \{\ \Lambda \text{ is an isomorphism}\ \} \\
& \Lambda X \cdot \mathsf{in} = \Lambda[\bot, S] \\
= & \quad \{\ \text{coproduct power-transpose (2.143)}\ \} \\
& \Lambda X \cdot \mathsf{in} = [\Lambda\bot, \Lambda S] \\
= & \quad \{\ \text{use (2.141) as } (id \times \Lambda X) \text{ is a function}\ \} \\
& \Lambda X \cdot \mathsf{in} = [\Lambda\bot, \Lambda T \cdot (id \times \Lambda X)] \\
= & \quad \{\ \mathbb{F}\ X = id + id \times X\ \} \\
& \Lambda X \cdot \mathsf{in} = [\Lambda\bot, \Lambda T] \cdot \mathbb{F}\ \Lambda X \\
= & \quad \{\ \text{cata}\ \} \\
& \Lambda X = (\!|\, [\Lambda\bot, \Lambda T] \,|\!)
\end{array}
$$

It remains to implement ΛT. With the power transpose laws for relational division (2.239) and intersection (2.238), we have:

$$\Lambda(R \,/\, \in^\circ)\ x = \{b \mid \langle \forall\ c\ :\ c \in x :\ b\ R\ c \rangle\}$$
$$\Lambda(\in \cdot \pi_2 \cap R \cdot \pi_1)\ (a, x) = \{b \mid b \in x \wedge b\ R\ a\}$$

The set resulting from the transpose of the division can be calculated because, by intersecting with π_1, we limit the set of values that can belong to the set to a single one. Joining the two sets, we get:

$$\begin{aligned}&\Lambda T\ (x, xs) = y \cup \{x' \mid x' \in xs \wedge x'\ R\ x\}\\ &\quad \textbf{where}\\ &\qquad y = \textbf{if}\ \langle \forall\, x' \ :\ x' \in xs :\ x\ R\ x' \rangle\ \textbf{then}\ \{x\}\ \textbf{else}\ \{\,\}\end{aligned}$$

Finally, we implement the *shrink* function, satisfying

$$\Lambda(\in \restriction R)\ \supseteq\ shrink\ R \tag{2.263}$$

by representing sets as finite lists, and carrying out set union inside the branches of the **if** expression:

$$\begin{aligned}&shrink\ r = (\!|\, [[\,], step] \,|\!)\ \textbf{where}\\ &\quad step\ (x, xs) =\\ &\qquad \textbf{let}\ xs' = filter\ (\lambda a \to r\ a\ x)\ xs\\ &\qquad \textbf{in if}\ all\ (\lambda a \to r\ x\ a)\ xs\\ &\qquad\quad \textbf{then}\ x : xs'\\ &\qquad\quad \textbf{else}\ xs'\end{aligned}$$

Unlike the *minlist* function of (Bird and de Moor, 1997), this implementation does not require R to be a connected preorder, it is sufficient for R to be reflexive. However, it has $O(n^2)$ time complexity in the worst case, whereas *minlist* is linear. Since, in practice, the input of *shrink* is generally a very reduced list of candidate solutions, the impact of this inefficiency is small.

2.8 SUMMARY

This chapter covered some theoretical aspects useful to the work ahead. Specifically, after going over basic concepts of relation algebra, three main constructs were introduced: the shrinking operator, crucial in specifying optimization processes; recursive relations, namely, catamorphisms and hylomorphisms, and their associated properties; and useful theorems for solving optimization problems, mostly adapted from (Bird and de Moor, 1997).

Part II

CONTRIBUTION

3

INDUCTIVE RELATIONS ON LISTS

Lists are among the simplest and most useful collection datatypes. This work addresses problems which make much use of them. In this chapter, the finite list datatype is defined, followed by useful inductive relations on lists: permutations, subsequences and sublists. Finally, some useful monotonicity rules are given.

3.1 THE LIST DATATYPE

This section defines the finite list datatype in relation to an initial algebra and its base relator, and describes some basic laws to manipulate lists in a relational context. We define the base binary relator $\mathbb{B}$ of finite lists, from which the previously defined functor $\mathbb{F}$ can be derived:

$$\mathbb{B}\ (X, Y) = 1 + X \times Y \tag{3.1}$$

Recalling the initial algebra $A \xleftarrow{\mathsf{in}} \mathbb{F}\ A$

$$\mathsf{in} = [nil, cons]$$

we aim to prove useful property of these functions. The membership relation on lists (ϵ), first introduced in the section on relational hylomorphisms, satisfies the equation

$$\epsilon \cdot \mathsf{in} = [\bot, \pi_1 \cup \epsilon \cdot \pi_2]$$

allowing us to state:

$$\epsilon \cdot nil = \bot \tag{3.2}$$

$$\epsilon \cdot cons = \pi_1 \cup \epsilon \cdot \pi_2 \tag{3.3}$$

In the next sections, a few important results involving membership will be proven using this equation.

A common operation on lists is to apply an operation to each element, preserving its initial order in the list — the map operation:

$$R^* = (\!|\, \mathsf{in} \cdot \mathbb{B}\ (R, id)\, |\!) \tag{3.4}$$

When R is a function f, f^* is also a function, and is injective or surjective if and only if f is injective or surjective, respectively. A useful property of the $cons$ function, called its 'free theorem' for the way it can be derived automatically from its type, involves this operation:

$$cons \cdot (R \times R^*) \subseteq R^* \cdot cons \tag{3.5}$$

What follows is a collection of laws and definitions used in proofs throughout this work when dealing with finite lists:

$$null = true \cdot nil^\circ \cup false \cdot cons^\circ \tag{3.6}$$
$$Hd = \pi_1 \cdot cons^\circ \tag{3.7}$$
$$Tl = \pi_2 \cdot cons^\circ \tag{3.8}$$
$$\phi_{null} = nil \cdot nil^\circ \tag{3.9}$$
$$\phi_{\neg\, null} = cons \cdot cons^\circ \tag{3.10}$$
$$\phi_{null} \cdot nil = nil \tag{3.11}$$

3.2 PERMUTATIONS

A permutation of a list is a list containing the same elements with the same multiplicity for each element. A concise relational definition, given in (Bird and de Moor, 1997), is the metaphorism

$$Perm = \frac{bag}{bag} \tag{3.12}$$

where bag, the multiset containing all elements of a list and their multiplicities, can be defined as a higher-order catamorphism over lists giving the function which calculates the multiplicity of an element in the list[1]. The definition simply states that any two permutations have the same bag of elements. As a consequence of this definition, we have that

$$Perm = Perm^\circ \tag{3.13}$$

and

$$Perm \cdot cons = Perm \cdot cons \cdot (id \times Perm) \tag{3.14}$$

which follows from Theorem 2 of (Oliveira, 2018) and the fact that a metaphorism $\frac{f}{\overline{f}}$ (where — recall — f is a fold) is a $\mathbb{F}$-congruence on the initial algebra of f (2.255). From this we can also state that the only permutation of the empty list is itself:

$$Perm \cdot nil = nil \tag{3.15}$$

1 As this definition is rarely needed, it is only given in (B.1), where it is most useful.

$$Perm \cdot \phi_{null} = \phi_{null} \tag{3.16}$$

The free theorem of $Perm$

$$Perm \cdot R^* \subseteq R^* \cdot Perm \tag{3.17}$$

can be strengethed to an equality when R is a function f:

$$f^* \cdot Perm = Perm \cdot f^* \tag{3.18}$$

Proof:

$$\begin{array}{ll} & f^* \cdot Perm = Perm \cdot f^* \\ \Leftarrow & \quad \{ \text{ free theorem of } Perm \text{ (3.17) leaves only } \} \\ & f^* \cdot Perm \subseteq Perm \cdot f^* \\ \equiv & \quad \{ \text{ shunting (2.58, 2.59) } \} \\ & Perm \cdot (f^*)^\circ \subseteq (f^*)^\circ \cdot Perm \\ \equiv & \quad \{ \text{ converses; } Perm = Perm^\circ \text{ (3.13) } \} \\ & Perm \cdot f^* \subseteq f^* \cdot Perm \\ \equiv & \quad \{ \text{ free theorem of } Perm \text{ (3.17) } \} \\ & f^* \cdot Perm \subseteq f^* \cdot Perm \\ \Box & \end{array}$$

SELECTING ELEMENTS Let us define the $Select$ relation that picks an element from a given list, producing pairs of elements and corresponding lists which can be combined to form a permutation of the original one. The $Discard$ relation ignores the selected element, giving a list with one fewer element:

$$Select = cons^\circ \cdot Perm \tag{3.19}$$

$$Discard = \pi_2 \cdot Select = \pi_2 \cdot cons^\circ \cdot Perm \tag{3.20}$$

A useful result states that selecting an element from a list and immediately adding it at the head yields a permutation:

$$cons \cdot Select \subseteq Perm \tag{3.21}$$

(Just shunt $cons$ in (3.19) to the left.) Moreover, if a list is non empty, then there is a selection that is neutralized by $cons$:

$$\phi_{\neg\, null} \subseteq cons \cdot Select \tag{3.22}$$

Proof:

$$
\begin{array}{ll}
 & \phi_{\neg\, null} \subseteq cons \cdot Select \\
\equiv & \quad \{ \text{ definition of } Select \ \} \\
 & \phi_{\neg\, null} \subseteq cons \cdot cons^{\circ} \cdot Perm \\
\equiv & \quad \{ \text{ (3.10); monotonicity (2.31) } \} \\
 & id \subseteq Perm \\
\equiv & \quad \{ \ Perm \text{ is reflexive (2.35) } \} \\
 & true \\
\end{array}
$$

□

Select "absorbs" permutations on either side of it:

$$Select \cdot Perm = Select = (id \times Perm) \cdot Select \tag{3.23}$$

The first equality is immediate from the fact $Perm \cdot Perm = Perm$ (because $Perm$ is both transitive and reflexive, i.e. a preorder). The proof of the second equality also demonstrates the upside of defining $Select$ in terms of $cons$ and $Perm$:

$$
\begin{array}{ll}
 & Select \\
= & \quad \{ \text{ definition of } Select \text{ (3.19) } \} \\
 & cons^{\circ} \cdot Perm \\
= & \quad \{ \text{ converse; } Perm = Perm^{\circ} \text{ (3.13) } \} \\
 & (Perm \cdot cons)^{\circ} \\
= & \quad \{ \text{ (3.14) } \} \\
 & (Perm \cdot cons \cdot (id \times Perm))^{\circ} \\
= & \quad \{ \text{ converse; } Perm = Perm^{\circ} \text{ (3.13) } \} \\
 & (id \times Perm) \cdot cons^{\circ} \cdot Perm \\
= & \quad \{ \text{ definition of } Select \text{ (3.19) } \} \\
 & (id \times Perm) \cdot Select \\
\end{array}
$$

□

Since *Select* is only defined for non-empty lists, the following equality holds:

$$Select \cdot \phi_{\neg\, null} = Select \tag{3.24}$$

Additionally, we can be sure that any selected element is a member of the list, and that the result of discarding an element leaves a smaller list:

$$\pi_1 \cdot Select \subseteq \epsilon \tag{3.25}$$

$$\epsilon \cdot Discard \subseteq \epsilon \tag{3.26}$$

The related $SelectBy\ R$ relation

$$SelectBy\ R = Select \upharpoonright R_{\pi_1} \tag{3.27}$$

which selects elements based on some order R, can be refined to a function if R is connected:

$$SelectBy\ R \ \supseteq\ selectMin\ R = \langle \pi_1, \ominus \rangle \cdot \langle minlist\ R, id \rangle \ \Leftarrow\ R \text{ is connected} \tag{3.28}$$

where $A^* \xleftarrow{\ominus} A \times A^*$ is the function which, given an element and a list, removes the first occurrence of the element in the list. The proof of this statement, including accompanying definitions, is laid out in Appendix B.

$Perm$ AS AN ANAMORPHISM We can also express $Perm$ in terms of the $Select$ relation, specifically as an anamorphism

$$Perm = (\!|\, T^\circ \,|\!)^\circ$$

for some T of type $\mathbb{F}\, A \xleftarrow{T} A$. Proof:

$$\begin{aligned}
& Perm = (\!|\, T^\circ \,|\!)^\circ \\
\equiv & \quad \{ \text{ converses ; } Perm \text{ is symmetric } \} \\
& Perm = (\!|\, T^\circ \,|\!) \\
\equiv & \quad \{ \text{ let } T := [T_1, T_2]^\circ \ \} \\
& Perm \cdot [nil, cons] = [T_1, T_2] \cdot (id + id \times Perm) \\
\equiv & \quad \{ \text{ choose } T_1 := nil \ \} \\
& Perm \cdot cons = T_2 \cdot (id \times Perm) \\
\equiv & \quad \{ \text{ converses } \} \\
& Select = (id \times Perm) \cdot T_2^\circ \\
\equiv & \quad \{ \text{ choose } T_2 := Select^\circ \ \} \\
& Select = (id \times Perm) \cdot Select \\
\equiv & \quad \{ \text{ (3.23) } \} \\
& true
\end{aligned}$$

□

And so we have successfully proven that $Perm$ is the anamorphism:

$$Perm = (\!|\, [nil, Select^\circ] \,|\!)^\circ \tag{3.29}$$

Since nil and $Select^\circ$ have disjoint ranges, it is possible to obtain an alternative representation of T, which will ultimately be easier to manipulate into a functional implementation.

$$T = (nil + Select) \cdot null? \tag{3.30}$$

Proof:

$$\begin{aligned}
& [nil, Select^\circ]^\circ \\
= & \quad \{ \text{ (3.11); (3.24), converse } \} \\
& [\phi_{null} \cdot nil, \phi_{\neg\, null} \cdot Select^\circ]^\circ \\
= & \quad \{ \text{ direct sum absorption (2.87) } \} \\
& [\phi_{null}, \phi_{\neg\, null}] \cdot (nil^\circ + Select)^\circ \\
= & \quad \{ \text{ converse } \} \\
& (nil + Select) \cdot [\phi_{null}, \phi_{\neg\, null}]^\circ \\
= & \quad \{ \text{ definition of guard (2.169) } \} \\
& (nil + Select) \cdot null? \\
& \square
\end{aligned}$$

With this, we can prove the following result, stating that any member of a list is a member of any of its permutations:

$$\epsilon \cdot Perm = \epsilon \tag{3.31}$$

Proof: see Appendix B.

STABLE FUNCTIONS A so-called *stable*[2] function f is one that produces the same value for all permutations of its input:

$$f \cdot Perm = f \quad \equiv \quad f \text{ is stable} \tag{3.32}$$

Because of this, any relation defined on the value of f, i.e. R_f, will relate two lists if and only if all permutations of those lists are themselves related.

$$Perm \cdot R_f = R_f = R_f \cdot Perm \tag{3.33}$$

2 This terminology is taken from (Korte and Lovász, 1984).

It suffices to prove one of these equalities to show how (3.32) is required by all of them:

$$
\begin{array}{ll}
 & Perm \cdot R_f \\
= & \quad \{ \text{ abbreviation (2.6) } \} \\
 & Perm \cdot f^\circ \cdot R \cdot f \\
= & \quad \{ \text{ converse; } Perm = Perm^\circ \ \} \\
 & (f \cdot Perm)^\circ \cdot R \cdot f \\
= & \quad \{ \ f \text{ is stable (3.32); abbreviation (2.6) } \} \\
 & R_f \\
\Box &
\end{array}
$$

If a predicate $Bool \xleftarrow{p} A^*$ is stable, then a coreflexive on that predicate has the following property:

$$\phi_p \cdot Perm = Perm \cdot \phi_p \tag{3.34}$$

Proof: see Appendix B.

3.3 SUBSEQUENCES AND SUBLISTS

This section covers two important preorders on lists: the subsequence ($\sqsubseteq$) and sublist ($\trianglelefteq$) relations. These will help specify optimization problems and derive functional implementations using their definitions and properties.

SUBSEQUENCES A subsequence of a list is a smaller list containing only elements from the original list, and so that the relative ordering of the elements is maintained. The subsequence relation, ($\sqsubseteq$), is the relational catamorphism

$$(\sqsubseteq) = (\!|\, [nil, cons \cup \pi_2] \,|\!)$$

which, with each new element, chooses whether to include or exclude from the resulting list. By the definition of a catamorphism (2.204), we have:

$$(\sqsubseteq) \cdot nil = nil \tag{3.35}$$

$$(\sqsubseteq) \cdot cons = (cons \cup \pi_2) \cdot (id \times (\sqsubseteq)) \tag{3.36}$$

With regards to membership, ($\sqsubseteq$) produces a list with fewer elements, so any element of a subsequence is an element of the full list:

$$\epsilon \cdot (\sqsubseteq) \subseteq \epsilon \tag{3.37}$$

Proof: see Appendix B.

We will need laws that state how the $Select$ and $Discard$ relations interact with $(\sqsubseteq)$. The relation that selects an element from a subsequence is same as one that selects an element and produces subsequences of the resulting list:

$$Select \cdot (\sqsubseteq) = (id \times (\sqsubseteq)) \cdot Select \tag{3.38}$$

Proof: see Appendix B.

The subsequence of a list from which an element has been discarded is a permutation of a subsequence of the original list:

$$(\sqsubseteq) \cdot Discard \subseteq Perm \cdot (\sqsubseteq) \tag{3.39}$$

Proof: see Appendix B.

Some basic properties can be observed about whether a list is empty or non-empty, which is expressed through relational statements involving ϕ_{null} and $\phi_{\neg\, null}$. The only subsequence of the empty list is the empty list itself:

$$(\sqsubseteq) \cdot \phi_{null} = \phi_{null} \tag{3.40}$$

Proof:

$$\begin{array}{ll} & (\sqsubseteq) \cdot \phi_{null} \subseteq \phi_{null} \\ \equiv & \quad \{\ \phi_{null} = nil \cdot nil^{\circ}\ (3.9)\ \} \\ & (\sqsubseteq) \cdot nil \cdot nil^{\circ} \subseteq \phi_{null} \\ \equiv & \quad \{\ (\sqsubseteq) \cdot nil = nil\ (3.35)\ \} \\ & nil \cdot nil^{\circ} \subseteq \phi_{null} \\ \equiv & \quad \{\ \phi_{null} = nil \cdot nil^{\circ}\ (3.9)\ \} \\ & true \\ \Box & \end{array}$$

Going in the other direction, if a subsequence of a list is non-empty, then the original list is also non-empty:

$$\phi_{\neg\, null} \cdot (\sqsubseteq) \subseteq (\sqsubseteq) \cdot \phi_{\neg\, null} \tag{3.41}$$

Finally, we prove that $(\sqsubseteq)$ is $\mathbb{F}$-compatible for $\mathbb{F}\ X = id + id \times X$

$$(\sqsubseteq) \xleftarrow{\mathsf{in}} \mathbb{F}\ (\sqsubseteq) \tag{3.42}$$

which unfolds into two conditions:

$$nil \subseteq (\sqsubseteq) \cdot nil$$

$$cons \cdot (id \times (\sqsubseteq)) \subseteq (\sqsubseteq) \cdot cons$$

The first condition is immediate from (3.35). The second is almost as simple:

$$\begin{array}{ll} & cons \cdot (id \times (\sqsubseteq)) \subseteq (\sqsubseteq) \cdot cons \\ \equiv & \quad \{ \text{ (3.36) } \} \\ & cons \cdot (id \times (\sqsubseteq)) \subseteq (cons \cup \pi_2) \cdot (id \times (\sqsubseteq)) \\ \Leftarrow & \quad \{ \text{ monotonicity (2.31) } \} \\ & cons \cdot (id \times (\sqsubseteq)) \subseteq cons \cdot (id \times (\sqsubseteq)) \\ \Box & \end{array}$$

SUBLISTS A sublist can be defined as a subsequence of any permutation of the original list, since the relative ordering of elements need not be fixed here.

$$(\trianglelefteq) = (\sqsubseteq) \cdot Perm = (\!|\, [nil, cons \cup \pi_2] \,|\!) \cdot (\!|\, [nil, Select^\circ] \,|\!)^\circ \tag{3.43}$$

This definition automatically gives us $(\trianglelefteq)$ as a hylomorphism, where the intermediate structure is a permutation, and the anamorphism and catamorphism genes represent the two phases of the process: non-deterministically selecting an element from the list, and then producing all sublists containing and not containing that element in that position. The following diagram represents the recursion scheme of the hylomorphism:

$$\begin{array}{ccc} A^* & \xleftarrow{[nil, Select^\circ]} & 1 + A \times A^* \\ {\scriptstyle (\trianglelefteq)}\downarrow & & \downarrow{\scriptstyle \mathbb{F}\ (\trianglelefteq)} \\ A^* & \xleftarrow[[nil, cons \cup \pi_2]]{} & 1 + A \times A^* \end{array}$$

Let us prove that $(\trianglelefteq)$ is $\mathbb{F}$-compatible (on finite lists). From

$$(\trianglelefteq) \xleftarrow{\text{in}} \mathbb{F}\ (\trianglelefteq) \tag{3.44}$$

two conditions arise:

$$nil \subseteq (\trianglelefteq) \cdot nil \tag{3.45}$$

$$cons \cdot (id \times (\trianglelefteq)) \subseteq (\trianglelefteq) \cdot cons \tag{3.46}$$

The first condition is trivially true from (3.15) and (3.35). The second condition follows from $Perm$ being a congruence and $(\sqsubseteq)$ being $\mathbb{F}$-compatible itself:

$$cons \cdot (id \times (\trianglelefteq))$$

$$
\begin{array}{ll}
= & \{ \text{ definition of } (\trianglelefteq);\ \times \text{ is a bifunctor } \} \\
& cons \cdot (id \times (\sqsubseteq)) \cdot (id \times Perm) \\
\subseteq & \{ \ (\sqsubseteq) \text{ is } \mathbb{F}\text{-compatible (3.42)} \ \} \\
& (\sqsubseteq) \cdot cons \cdot (id \times Perm) \\
\subseteq & \{ \ Perm \text{ is a congruence (3.14)} \ \} \\
& (\sqsubseteq) \cdot Perm \cdot cons \\
= & \{ \text{ definition of } (\trianglelefteq) \ \} \\
& (\trianglelefteq) \cdot cons \\
\Box &
\end{array}
$$

An alternate definition of $(\trianglelefteq)$ is as a permutation of a subsequence, meaning that a subsequence of a permutation is a permutation of a subsequence, and vice-versa:

$$(\sqsubseteq) \cdot Perm = Perm \cdot (\sqsubseteq) \tag{3.47}$$

See the proof in Appendix B.

3.4 MONOTONICITY RULES

Establishing monotonicity with respect to a preorder is an important step in refining relational specifications of optimization problems. For finite lists, we wish to establish monotonicity of the initial algebra in, so that we may prove results where element are added to a list. For some element type A, and taking $\mathbb{F}$ to be the functor for finite lists ($\mathbb{F}\ X = 1 + A \times X$) and $\mathbb{G}\ A = A \times X$, we have that:

$$R \xleftarrow{\mathsf{in}} \mathbb{F}\ R \quad \equiv \quad \begin{cases} R \xleftarrow{nil} id \\ R \xleftarrow{cons} \mathbb{G}\ R \end{cases} \tag{3.48}$$

Proof:

$$
\begin{array}{ll}
& R \xleftarrow{\mathsf{in}} \mathbb{F}\ R \\
\equiv & \{ \text{ definition (2.132) } \} \\
& \mathsf{in} \cdot \mathbb{F}\ R \subseteq R \cdot \mathsf{in} \\
\equiv & \{ \ \mathsf{in} = [nil, cons], \text{ coproducts (2.90) } \} \\
& \begin{cases} nil \subseteq R \cdot nil \\ cons \cdot \mathbb{G}\ R \subseteq R \cdot cons \end{cases} \\
\equiv & \{ \text{ definition (2.132) (twice) } \}
\end{array}
$$

$$\begin{cases} R \xleftarrow{nil} id \\ R \xleftarrow{cons} \mathbb{G}\ R \end{cases}$$

□

For preorders evaluated on a function h, written as R_h, we may instantiate h to obtain specific more laws. With $h := f^*$, we rely on the free theorem of $cons$ (3.5) to get a sufficient condition for monotonicity (under the $\mathbb{G}$ functor):

$$R_{f^*} \xleftarrow{cons} \mathbb{G}\ R_{f^*} \quad \Leftarrow \quad R \xleftarrow{cons} \mathbb{G}\ R \tag{3.49}$$

For $h := (\!|\, g \,|\!)$, a functional catamorphism, we infer monotonicity on a fold by monotonicity on its gene, g:

$$S_{(\!|g|\!)} \xleftarrow{\mathsf{in}} \mathbb{F}\ S_{(\!|g|\!)} \quad \Leftarrow \quad S \xleftarrow{g} \mathbb{F}\ S \tag{3.50}$$

The proofs of (3.49) and (3.50) are present in Appendix B.

Let us use the above laws to to prove monotonicity of $cons$ with respect a preorder evaluated on a linear objective function $h := sum \cdot f^*$:

$$(\geqslant)_{sum \cdot f^*} \xleftarrow{cons} \mathbb{G}\ (\geqslant)_{sum \cdot f^*} \tag{3.51}$$

We reason:

$$\begin{array}{ll} & (\geqslant)_{sum \cdot f^*} \xleftarrow{cons} \mathbb{G}\ (\geqslant)_{sum \cdot f^*} \\ \Leftarrow & \quad \{\ R_{f \cdot g} = S_g \text{ where } S = R_f, (3.49)\ \} \\ & (\geqslant)_{sum} \xleftarrow{cons} \mathbb{G}\ (\geqslant)_{sum} \\ \Leftarrow & \quad \{\ (3.48)\ \} \\ & (\geqslant)_{sum} \xleftarrow{\mathsf{in}} \mathbb{F}\ (\geqslant)_{sum} \\ \Leftarrow & \quad \{\ sum = (\!|\, [\underline{0}, add] \,|\!)\ (3.50)\ \} \\ & (\geqslant) \xleftarrow{[\underline{0}, add]} \mathbb{F}\ (\geqslant) \\ \equiv & \quad \{\ \text{definition (2.132)}\ \} \\ & [\underline{0}, add] \cdot \mathbb{F}\ (\geqslant) \subseteq (\geqslant) \cdot [\underline{0}, add] \\ \equiv & \quad \{\ \text{coproducts (2.90)}\ \} \\ & \begin{cases} \underline{0} \subseteq (\geqslant) \cdot \underline{0} \\ add \cdot (id \times (\geqslant)) \subseteq (\geqslant) \cdot add \end{cases} \\ \Leftarrow & \quad \{\ \text{monotonicity (2.27); } (\geqslant) \text{ is reflexive (2.35), monotonicity (2.32)}\ \} \\ & add \cdot ((\geqslant) \times (\geqslant)) \subseteq (\geqslant) \cdot add \end{array}$$

$\equiv$ $\{$ monotonicity of addition with respect to $(\geqslant)$ $\}$

$true$

$\square$

3.5 SUMMARY

In this chapter, the finite list datatype was introduced and a few associated relations were defined. The permutation relation $Perm$ was proven to be a congruence which can be expressed as an anamorphism. Sublists were defined as subsequences of permutations — $(\sqsubseteq) \cdot Perm$ — or vice-versa, which means the sublist relation $(\trianglelefteq)$ is $\mathbb{F}$-compatible. Finally, some useful monotonicity rules for preorders of the form R_f were given.

4

THE GREEDY ALGORITHM

This chapter proposes a general form for the greedy algorithm, explores its the algorithmic properties through the theory of matroids, and then uses a simple case study to demonstrate these ideas.

The greedy algorithm is an algorithm which makes locally optimal choices to arrive at a globally optimal solution. The most important aspect of its operation is the fact that only a single solution is considered at each step. It is up to the algorithm designer to find a criterion of local optimality for the given problem. They need to prove that, at each step, the constructed solution using that criterion is optimal considering only the part of the input so far processed, and so, when the whole input is processed, the complete solution is guaranteed to be optimal.

In imperative programming languages, the greedy algorithm can be implemented using a while loop. Starting with the empty solution, parts of the input are considered in descending order of usefulness according to the local optimality criterion. Then, the solution is successively augmented with each considered part if it results in a feasible solution, that is, one that obeys the restrictions set by the problem specification. In the end, an optimal solution is obtained for the entire input.

4.1 A GENERAL FORM

In this section, a general form for the greedy algorithm will be proposed, implementing the imperative-style while loop in a relational context. This will allow for a more calculational approach to deriving the greedy algorithm.

In essence, the while loop implementing the greedy algorithm is a two-phase process: a locally optimal choice is made, and then the current solution is augmented based on such choice. This suggests a hylomorphism as a possible general form for the greedy algorithm, segregating the two types of decisions into the two phases of the hylomorphism. The divide phase makes the locally optimal choice to determine the order by which the input is considered, while the conquer phase constructs the optimal solution by processing the virtual data structure in order.

Out of the known shrinking and thinning theorems in the literature, the Greedy Theorem (Theorem 2.2) deals only with catamorphisms, while others involving hylomorphisms (Theorems 2.3, 2.4, 2.5) rely on strict monotonicity conditions.

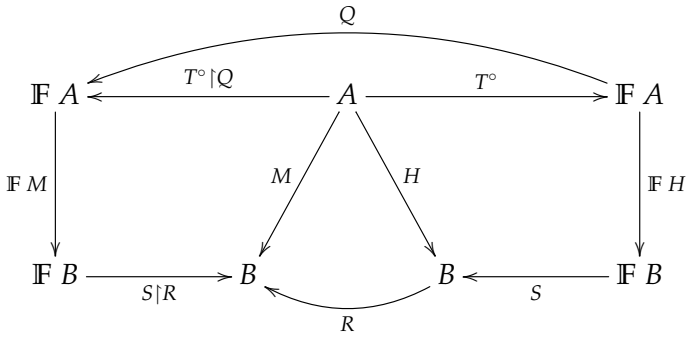
The following fixpoint equation,

$$\langle \mu\, X \;::\; (S \upharpoonright R) \cdot \mathbb{F}\, X \cdot (T \upharpoonright Q) \rangle \subseteq M \tag{4.1}$$

is meant to be a starting point from which to derive the greedy algorithm for specific problems, where

- $M = H \upharpoonright R$ and $H = (\!|\, S \,|\!) \cdot (\!|\, T^\circ \,|\!)^\circ$;
- Q is a preorder representing the criterion for making an 'initial choice';
- shrinking is to be performed in both the divide and the conquer phases of the hylomorphism.

The following type diagram illustrates the structure of the two hylomorphisms in play, and how they interact.



Note that, due to the use of shrinking, there is only one solution under consideration in each phase. The challenge lies in obtaining an appropriate definition of Q, such that the final optimization ($\upharpoonright R$) can be made efficiently and the hylomorphism can be refined to a functional implementation.

4.2 MATROIDS AND GREEDOIDS

In this section, matroids are defined in terms of set systems and the notion of feasibility[1] over a set system. Then, some important properties are described and expressed in relational algebra. With no loss of generality, we represent sets as finite lists when expressing them.

SET SYSTEMS A *set system* (S, C) consists of a base set of elements S, and a collection C of subsets of S, which defines the notion of feasibility. A set X is *feasible* if and only if $X \in C$. This set can be defined by a feasibility condition p, meaning that a set is feasible if and only if $p\ X$ is true.

A set system is *accessible* if it obeys the trivial axiom — the empty set (list) is feasible —

$$\phi_p \cdot nil = nil \tag{4.2}$$

1 This terminology was chosen over 'independence', as seen in some of the literature, due to being conceptually similar to restricting the solution space in optimization problems.

and the accessibility axiom: if a non-empty set X is feasible, then $\exists\ x \in X$ such that $X - \{x\}$ is feasible. A *hereditary* set system is one in which every subset of a feasible set is also feasible:

$$(\trianglelefteq) \cdot \phi_p \subseteq \phi_p \cdot \top \tag{4.3}$$

In the sequel, a weaker property will be useful,

$$\phi_p \cdot cons \subseteq cons \cdot (id \times \phi_p) \tag{4.4}$$

because it leads to the following equality

$$\phi_p \cdot cons = \phi_p \cdot cons \cdot (id \times \phi_p) \tag{4.5}$$

whose proof can be found in Appendix B.

A *matroid*, then, is a hereditary set system that obeys the augmentation axiom: if X and Y are feasible sets and $|X| = |Y| + 1$ then $\exists\ x \in X - Y$ such that $Y \cup \{x\}$ is feasible. A *greedoid* is a relaxation of a matroid: it obeys the augmentation axiom, but it need not be a hereditary set system. In the rest of the chapter, a matroidal structrure is assumed.

GREEDY POST-CONDITIONING In general, for any algebra $A \xleftarrow{S} \mathbb{F}\ A$ of a container structure of type A such that (generic) feasibility holds for "smaller" structures

$$\phi_p \cdot S \subseteq S \cdot \mathbb{F}\ \phi_p \tag{4.6}$$

we can prove the following greedy post-conditioning rule:

$$\phi_p \cdot (\!|\, S \,|\!) = (\!|\, \phi_p \cdot S \,|\!) \tag{4.7}$$

Proof:

$$\begin{array}{ll} & \phi_p \cdot (\!|\, S \,|\!) = (\!|\, \phi_p \cdot S \,|\!) \\ \Leftarrow & \quad \{\ \text{cata fusion (2.216)}\ \} \\ & \phi_p \cdot S = \phi_p \cdot S \cdot \mathbb{F}\ \phi_p \\ \equiv & \quad \{\ \mathbb{F}\ \phi_p\ \subseteq\ id\text{; monotonicity}\ \} \\ & \phi_p \cdot S \subseteq \phi_p \cdot S \cdot \mathbb{F}\ \phi_p \\ \equiv & \quad \{\ \phi_p \cdot X\ \subseteq\ Y \equiv \phi_p \cdot X\ \subseteq\ \phi_p \cdot Y\ \} \\ & \phi_p \cdot S \subseteq S \cdot \mathbb{F}\ \phi_p \\ \equiv & \quad \{\ \text{assumption (4.6)}\ \} \\ & true \end{array}$$

□

For instance, for $S := [nil, cons \cup \pi_2]$ (the gene of $(\sqsubseteq)$), assumption (4.6) results in:

$$
\begin{array}{ll}
& \phi_p \cdot [nil, cons \cup \pi_2] \subseteq [nil, cons \cup \pi_2] \cdot \mathbb{F}\ \phi_p \\
\equiv & \quad \{ \text{ coproducts (2.90) } \} \\
& \left\{ \begin{array}{l} \phi_p \cdot nil \subseteq nil \\ \phi_p \cdot (cons \cup \pi_2) \subseteq (cons \cup \pi_2) \cdot (id \times \phi_p) \end{array} \right. \\
\Leftarrow & \quad \{ \text{ (4.2); join bilinearity (2.20), universal property (2.14); monotonicity (2.31) (twice) } \} \\
& \left\{ \begin{array}{l} \phi_p \cdot cons \subseteq cons \cdot (id \times \phi_p) \\ \phi_p \cdot \pi_2 \subseteq \pi_2 \cdot (id \times \phi_p) \end{array} \right. \\
\equiv & \quad \{ \text{ (4.5); Kronecker product cancellation (2.75) } \} \\
& true \\
\Box &
\end{array}
$$

This allows us to state

$$\phi_p \cdot (\!| [nil, cons \cup \pi_2] |\!) = (\!| [nil, \phi_p \cdot cons \cup \pi_2] |\!) \tag{4.8}$$

Proof:

$$
\begin{array}{ll}
& \phi_p \cdot (\!| [nil, cons \cup \pi_2] |\!) \\
= & \quad \{ \text{ (4.7) } \} \\
& (\!| \phi_p \cdot [nil, cons \cup \pi_2] |\!) \\
= & \quad \{ \text{ coproduct fusion (2.83); (4.2) } \} \\
& (\!| [nil, \phi_p \cdot (cons \cup \pi_2)] |\!) \\
= & \quad \{ \text{ claim: see page 131 } \} \\
& (\!| [nil, \phi_p \cdot cons \cup \pi_2] |\!) \\
\Box &
\end{array}
$$

GREEDY AUGMENTATION Now, from a non-deterministic augmentation relation $S = [nil, \phi_p \cdot cons \cup \pi_2]$ obtained from (4.8), we wish to derive a function by optimization (shrinking). And indeed we can: let $f = sum \cdot wt^*$, where weight function wt is positive-valued; by equational reasoning, we prove (in Appendix B, p. 132) that $S \upharpoonright R$, for $R = (\geqslant)_f$ is the function

$$S \upharpoonright R = [nil, aug] \tag{4.9}$$

where

$$aug = p \cdot cons \rightarrow cons, \pi_2 \tag{4.10}$$

is the function which augments a solution given a new element of the list, including it only if the larger list is feasible.

4.3 OPTIMALITY OF THE GREEDY ALGORITHM

Using the theory of matroids, generalized to greedoids, Korte and Lovász (1984) prove the correctness of the greedy algorithm for certain classes of problems. Specifically, they prove that an optimization problem with a greedoid as the underlying set system and an objective function — the function whose value we wish to optimize — satisfying a few *consistency* conditions, the greedy algorithm gives the optimal solution.

In the rest of this section, a few general statements about the greedy algorithm under a matroidal structure and a linear objective function, for some nonnegative weight function wt, will be proven. We assume the linear objective function

$$f = sum \cdot wt^*$$

in the definition of M,

$$M = \underbrace{\phi_p \cdot E}_{H} \upharpoonright \underbrace{(\geqslant)_f}_{R} \tag{4.11}$$

which optimizes over the solution space represented by $A^* \xleftarrow{E} A^*$ under some assumptions:

- E is restricted by feasibility condition $Bool \xleftarrow{p} A^*$, that is, only outputs of E satisfying this condition are considered for optimization;
- E is $\mathbb{F}$-compatible on finite lists, that is,

$$cons \cdot (id \times E) \subseteq E \cdot cons \tag{4.12}$$

  holds;
- H satisfies the following condition:

$$H \cdot cons = H \cdot cons \cdot (id \times H) \tag{4.13}$$

- E produces lists which are no better (according to R) than the input. That is, f has relational type $(\leqslant) \xleftarrow{f} E$, meaning:

$$f \cdot E \subseteq (\leqslant) \cdot f \tag{4.14}$$

Under these assumptions, which will have to be ensured for each specific problem, we are able to prove the following general statement:

> *If, after running the greedy algorithm to obtain a partial solution it is possible to augment it with a new element, then the resulting list is the optimal solution when considering the full list.*

In relational algebra, we state this as

$$\phi_p \cdot cons \cdot (id \times M) \subseteq M \cdot cons \tag{4.15}$$

and proceed with its proof:

$$\begin{array}{ll} & \phi_p \cdot cons \cdot (id \times M) \subseteq M \cdot cons \\ \equiv & \quad \{ \text{ (4.11); function shrinking (2.191) } \} \\ & \phi_p \cdot cons \cdot (id \times M) \subseteq H \cdot cons \upharpoonright R \\ \equiv & \quad \{ \text{ universal property of shrinking (2.185) } \} \\ & \left\{ \begin{array}{l} \phi_p \cdot cons \cdot (id \times M) \subseteq H \cdot cons \\ \phi_p \cdot cons \cdot (id \times M) \cdot (H \cdot cons)^\circ \subseteq R \end{array} \right. \end{array}$$

The first condition is straightforward:

$$\begin{array}{ll} & \phi_p \cdot cons \cdot (id \times M) \subseteq H \cdot cons \\ \Leftarrow & \quad \{ \text{ shrinking cancellation (2.187); monotonicity (2.32, 2.27) } \} \\ & cons \cdot (id \times E) \subseteq E \cdot cons \\ \equiv & \quad \{ \text{ (4.12) } \} \\ & true \\ \Box & \end{array}$$

The second condition relies on our assumptions and the fact that f is a linear objective function:

$$\begin{array}{ll} & \phi_p \cdot cons \cdot (id \times M) \cdot (H \cdot cons)^\circ \subseteq R \\ \Leftarrow & \quad \{ \text{ monotonicity (2.32) } \} \\ & cons \cdot (id \times M) \cdot (H \cdot cons)^\circ \subseteq R \\ \equiv & \quad \{ \text{ (4.13) } \} \\ & cons \cdot (id \times M) \cdot (H \cdot cons \cdot (id \times H))^\circ \subseteq R \\ \Leftarrow & \quad \{ \text{ converse; } \times \text{ is a bifunctor; shrinking cancellation (2.188) } \} \\ & cons \cdot (id \times R) \cdot cons^\circ \cdot H^\circ \subseteq R \\ \equiv & \quad \{ \text{ since } R = (\geqslant)_{sum \cdot v^*} \text{, we apply (3.51) } \} \\ & R \cdot cons \cdot cons^\circ \cdot H^\circ \subseteq R \end{array}$$

$$
\begin{array}{ll}
\Leftarrow & \{ \text{ monotonicity (2.32) (twice) } \} \\
& R \cdot E^{\circ} \subseteq R \\
\equiv & \{ \text{ } R \text{ is transitive (2.37), so it is enough to prove } \} \\
& E^{\circ} \subseteq R \\
\equiv & \{ \text{ converses; definition of } R \text{; shunting (2.58) } \} \\
& f \cdot E \subseteq (\leqslant) \cdot f \\
\equiv & \{ \text{ (4.14) } \} \\
& true \\
\Box &
\end{array}
$$

Another important statement relates to the order by which the solution is augmented. By picking the smallest element to add last and computing the optimal solution of the rest of the list first, then, if we are unable to add the last element to the resulting solution, it is guaranteed to be the optimal solution for the entire list too. In relation algebraic terms:

$$\delta \, (\phi_{\neg \, p} \cdot cons) \cdot (id \times M) \cdot (Select \upharpoonright (\leqslant)_{v \cdot \pi_1}) \subseteq \pi_2^{\circ} \cdot M \tag{4.16}$$

Informal proof:[2] suppose that having picked the smallest element to add last, we find that the solution computed on the rest of the list (which is necessarily feasible) is not the optimal one, and that adding the last element does not result in a feasible solution; then, the chosen element must be part of the optimal solution, and at least one element of the calculated one must be excluded; however, since the chosen element is the smallest, the total value of the solution cannot increase by any such substitution; nor can other elements from the original list be included as to increase the size of the solution, because the augmentation axiom would dictate that one of those, or the chosen element, could be added to the computed solution to get a feasible list, resulting in a contradiction (the chosen element cannot be added), or in a better computed solution, which is, by definition, not possible; this implies the calculated solution is, in fact, optimal, which is a contradiction.

4.4 THE COLOR PROBLEM

This section presents a case study[3] in the derivation of the greedy algorithm under a matroidal structure.

DESCRIPTION Consider a list of items S, each with a *measure* and a *dimension*. For the color problem, the measure is a nonnegative numeric value, and the dimension is the color of the item. Select a list of items maximizing the total value, while making sure that no two items have the same color, and that all the colors in the given list S can be found in the outcome.

2 See section 7.2 for more about this.
3 This particular instance of the problem was taken from (Guimarães, 2021).

To specify the problem relationally, we first specify the solution space, which, in this case, is made of all sublists of S:

$$E = (\trianglelefteq) = (\sqsubseteq) \cdot Perm$$

Then, with $c : Color \leftarrow Item$ being the function that tells the color of an item, we impose the post-condition that no colors can repeat in the output sets,

$$P = \phi_p \textbf{ where } p = noRepeats \cdot c^*$$

and the invariant stating the preservation of the color set:

$$I = \frac{Cs}{Cs} \textbf{ where } Cs = \epsilon \cdot c^*$$

Finally, we maximize by the objective function, where $v : Value \leftarrow Item$ is the weight function on items:

$$R = (\geqslant)_{value} \textbf{ where } value = sum \cdot v^*$$

These four parts combined give us a relational specification that can be suited to describe many problems:

$$M = (P \cdot E \cap I) \upharpoonright R \tag{4.17}$$

Consider the set system where the base set is the set of all possible items, and the feasible sets are those that don't contain items with the same color. This set system is a matroid: every subset contains fewer colors, therefore it is feasible if the original set is; and for two sets of different sizes, there is an item in the bigger set which we can add to the smaller one to form a feasible set, because at least one color will be missing from the smaller one.

Since we are dealing with a (stable) linear objective function and a matroidal structure, we know that the greedy algorithm gives the optimal solution. We now wish to calculationally prove this for the color problem, obtaining a functional implementation in the process.

Let us now state a few useful properties of $p = noRepeats \cdot c^*$ and $R = (\geqslant)_{value}$ themselves. Firstly, if a list has no two items with the same color, adding an element such that the resulting list repeats colors means that the new item's color is present in the original list. To express this in relational algebra, it is convenient to make use of coreflexives:

$$\begin{aligned}
& p\ xs\ \wedge\ \neg\ p\ (cons\ x\ xs) \Rightarrow \langle \exists\ a\ :\ a\ \epsilon\ xs :\ c\ x = c\ a \rangle \\
\Rightarrow \quad & \{\ \text{strengthen antecedent; introduce } (x, xs) \text{ as argument using } \pi_1 \text{ and } \pi_2\ \} \\
& y = (x, xs)\ \wedge\ (p \cdot \pi_2)\ (x, xs)\ \wedge\ (\neg\ p \cdot cons)\ (x, xs) \\
& \quad \Rightarrow \langle \exists\ a\ :\ a\ (\epsilon \cdot \pi_2)\ (x, xs) :\ (c \cdot \pi_1)\ y = c\ a \rangle \\
\equiv \quad & \{\ \text{convert to pointfree}\ \} \\
& (id \times \phi_p) \cdot \phi_{\neg\ p \cdot cons} \subseteq \frac{c}{c \cdot \pi_1} \cdot \epsilon \cdot \pi_2
\end{aligned}$$

$$\equiv \quad \{ \text{ symmetric division (2.112) } \}$$
$$(id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq (c \cdot \pi_1)^\circ \cdot c \cdot \epsilon \cdot \pi_2 \tag{4.18}$$

And since we are dealing with nonnegative numeric values, we can state that any list is better than its tail:

$$y = cons\ x\ xs \Rightarrow y\ R\ xs$$
$$\equiv \quad \{ \ \pi_2\ (x, xs) = xs \text{; convert to pointfree } \}$$
$$cons \subseteq R \cdot \pi_2 \tag{4.19}$$

DERIVATION By (3.43), E is a hylomorphism. The plan is to prove that $H = P \cdot E$ is also a hylomorphism, then obtain a candidate solution C by refining a modified version of the specification where the invariant I is missing

$$C \subseteq \underbrace{P \cdot E \upharpoonright R}_{M'}$$

and finally proving that C conforms to the original specification, i.e. $C \subseteq M$. To prove that $P \cdot E$ is a hylomorphism, we simply apply the greedy post-conditioning rule to fuse the post-condition P with the catamorphic part of E:

$$P \cdot E$$
$$= \quad \{ \text{ definitions of } P \text{ and } E \ \}$$
$$\phi_p \cdot (\sqsubseteq) \cdot Perm$$
$$= \quad \{ \text{ greedy post-conditioning for } (\sqsubseteq) \text{ (4.8) } \}$$
$$(\!|\, [nil, \phi_p \cdot cons \cup \pi_2] \,|\!) \cdot Perm$$

Now that we have expressed $P \cdot E$ as the hylomorphism (note the alternative definition of $Perm$ (3.30))

$$P \cdot E = \underbrace{(\!|\, [nil, \phi_p \cdot cons \cup \pi_2] \,|\!)}_{S} \cdot \underbrace{(\!|\, ((nil + Select) \cdot null?)^\circ \,|\!)^\circ}_{T} \tag{4.20}$$

we wish to obtain the greedy algorithm over the solution space it represents. Let us elaborate on (4.1) to obtain the criterion for the initial choice Q:

$$\langle \mu\ X\ ::\ (S \upharpoonright R) \cdot \mathbb{F}\ X \cdot (T^\circ \upharpoonright Q) \rangle \subseteq M'$$
$$\Leftarrow \quad \{ \text{ hylomorphism least fixpoint (2.220); converse } \}$$
$$(S \upharpoonright R) \cdot \mathbb{F}\ M' \cdot (T^\circ \upharpoonright Q) \subseteq M'$$
$$\equiv \quad \{ \text{ universal property of shrinking (2.185) } \}$$
$$\begin{cases} (S \upharpoonright R) \cdot \mathbb{F}\ M' \cdot (T^\circ \upharpoonright Q) \subseteq H \\ (S \upharpoonright R) \cdot \mathbb{F}\ M' \cdot (T^\circ \upharpoonright Q) \cdot H^\circ \subseteq R \end{cases}$$

$$
\begin{array}{ll}
\Leftarrow & \{ \text{ shrinking cancellation (2.187) (three times) } \} \\
& \left\{ \begin{array}{l} S \cdot \mathbb{F}\, H \cdot T^\circ \subseteq H \\ (S \upharpoonright R) \cdot \mathbb{F}\, M' \cdot (T^\circ \upharpoonright Q) \cdot H^\circ \subseteq R \end{array} \right. \\
\Leftarrow & \{ \text{ hylomorphism least fixpoint (2.220) } \} \\
& (S \upharpoonright R) \cdot \mathbb{F}\, M' \cdot (T^\circ \upharpoonright Q) \cdot H^\circ \subseteq R \\
\equiv & \{ \text{ (4.9); definition of } T\text{, converse } \} \\
& [nil, aug] \cdot \mathbb{F}\, M' \cdot ((nil + Select) \cdot null? \upharpoonright Q) \cdot H^\circ \subseteq R \\
\Leftarrow & \{ \text{ function shrinking (2.191); make } Q := id + U \text{ and promote shrinking into direct sum (2.202) } \} \\
& [nil, aug] \cdot \mathbb{F}\, M' \cdot (nil + Select \upharpoonright U) \cdot null? \cdot H^\circ \subseteq R \\
\equiv & \{ \text{ coproducts (2.90) } \} \\
& [nil, aug \cdot (id \times M') \cdot (Select \upharpoonright U)] \cdot null? \cdot H^\circ \subseteq R \\
\equiv & \{ \text{ biproduct absorption (2.85); bilinearity (2.21); universal property of join (2.14) } \} \\
& \left\{ \begin{array}{l} nil \cdot \phi_{null} \cdot H^\circ \subseteq R \\ aug \cdot (id \times M') \cdot (Select \upharpoonright U) \cdot \phi_{\neg\, null} \cdot H^\circ \subseteq R \end{array} \right.
\end{array}
$$

The first condition states that the empty list has a smaller or equal value than any of its sublists. This is true because the only sublist of the empty list is itself, and R is reflexive (2.35). The proof is shown in Appendix B. The second condition is divided in two statements by expanding the definition of aug, a McCarthy conditional (2.170), and the properties of join (2.21, 2.14):

$$
\left\{ \begin{array}{l} cons \cdot \phi_{p \cdot cons} \cdot (id \times M') \cdot (Select \upharpoonright U) \cdot H^\circ \subseteq R \\ \pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times M') \cdot (Select \upharpoonright U) \cdot H^\circ \subseteq R \end{array} \right.
$$

The first statement asks us to prove the correctness of the greedy algorithm when the smallest element is successfully added to the list:

$$
\begin{array}{ll}
& cons \cdot \phi_{p \cdot cons} \cdot (id \times M') \cdot (Select \upharpoonright U) \cdot H^\circ \subseteq R \\
\equiv & \{ \; f \cdot \phi_{p \cdot f} = \phi_p \cdot f \text{ (2.164) } \} \\
& \phi_p \cdot cons \cdot (id \times M') \cdot (Select \upharpoonright U) \cdot H^\circ \subseteq R \\
\Leftarrow & \{ \; \phi_p \cdot cons \cdot (id \times M') \subseteq M' \cdot cons \text{ (4.15) } \} \\
& M' \cdot cons \cdot (Select \upharpoonright U) \cdot H^\circ \subseteq R \\
\Leftarrow & \{ \text{ shrinking cancellation; } cons \cdot Select \subseteq Perm \text{ (3.21) } \} \\
& M' \cdot Perm \cdot H^\circ \subseteq R \\
\equiv & \{ \text{ converse; } H \cdot Perm = \phi_p \cdot (\sqsubseteq) \cdot Perm \cdot Perm = \phi_p \cdot (\sqsubseteq) \cdot Perm = H \; \} \\
& M' \cdot H^\circ \subseteq R
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{ \text{ shrinking cancellation (2.188) } \} \\
& true \\
\Box &
\end{array}
$$

To establish (4.15) for the color problem, we note that $(\trianglelefteq)$ is $\mathbb{F}$-compatible on finite lists (3.44), and verify the remaining conditions given earlier in this chapter:

$$\phi_p \cdot (\trianglelefteq) \cdot cons = \phi_p \cdot (\trianglelefteq) \cdot cons \cdot (id \times \phi_p \cdot (\trianglelefteq)) \tag{4.21}$$

$$sum \cdot v^* \cdot \phi_p \cdot (\trianglelefteq) \subseteq (\leqslant) \cdot sum \cdot v^* \tag{4.22}$$

The first essentially states that a feasible solution can be constructed incrementally — for any feasible solution, it is possible to construct a feasible solution from all elements but one, and then determine whether to include it or not. The second condition states that any feasible solution smaller in size will also be smaller in value. This is true because v does not produce negative values. The proof of these statements are present in Appendix B.

The second statement above relates to the case where it is not possible to augment the current solution using the element under consideration. This being the smallest element, it is assured that the optimal solution is given by the current one, calculated by the greedy algorithm up to that point:

$$
\begin{array}{ll}
& \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \cdot (id \times M') \cdot (Select \upharpoonright U) \cdot H^\circ \subseteq R \\
\Leftarrow & \{ \text{ apply shrinking cancellation (2.188) by proving } \} \\
& \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \cdot (id \times M') \cdot (Select \upharpoonright U) \subseteq M' \\
\equiv & \{ \text{ (2.178); shunting (2.58) } \} \\
& \delta\ (\phi_{\neg\ p} \cdot cons) \cdot (id \times M') \cdot (Select \upharpoonright U) \subseteq \pi_2^\circ \cdot M'
\end{array}
$$

Since $R = (\geqslant)_f$ where $f = sum \cdot v^*$, by making $U := (\leqslant)_{v \cdot \pi_1}$, we get (4.16). So we have

$$C = (\!|\, [nil, aug] \,|\!) \cdot [\![(nil + SelectBy\ (\leqslant)) \cdot null?]\!] \tag{4.23}$$

which can be shown to satisfy the complete specification including the invariant I:

$$C \subseteq \underbrace{(P \cdot E \cap I) \upharpoonright R}_{M}$$

Please see Appendix B, p. 144 for details.

We finish the calculation of $T^\circ \upharpoonright Q$ by exploiting the fact we are dealing with a connected preorder:

$$
\begin{array}{ll}
& T^\circ \upharpoonright Q \\
= & \{ \ T^\circ = (nil + Select) \cdot null? \ \} \\
& (nil + Select) \cdot null? \upharpoonright Q \\
= & \{ \text{ function shrinking (2.191) } \}
\end{array}
$$

$$
\begin{array}{ll}
 & ((nil + Select) \upharpoonright Q) \cdot null\,? \\
= & \qquad \{ \text{ make } Q = id + (\leqslant)_{v \cdot \pi_1} \text{; direct sum shrinking (2.202); function shrinking (2.190) } \} \\
 & (nil + Select \upharpoonright (\leqslant)_{v \cdot \pi_1}) \cdot null\,? \\
\supseteq & \qquad \{ \ (\leqslant) \text{ is a connected preorder, so apply (3.28) } \} \\
 & (nil + selectMin\ (\leqslant)_v) \cdot null?
\end{array}
$$

And so:

$$C \ \supseteq \ (\!|\, [nil, aug] \,|\!) \cdot [\!(nil + selectMin\ (\leqslant)_v) \cdot null?)\!]$$

In Haskell:

$$
\begin{array}{l}
p\ l = \mathsf{length}\ l \equiv \mathsf{length}\ (nubBy\ (\lambda x\ y \to c\ x \equiv c\ y)\ l) \\
aug = p \cdot cons \to cons, \pi_2 \\
selectMin\ r = \langle \pi_1, \widehat{delFirst} \rangle \cdot \langle minlist\ r, id \rangle \\
colors = hyloList\ g\ h\ \mathbf{where} \\
\quad h = (nil + selectMin\ (\leqslant)_v) \cdot null? \\
\quad g = [nil, aug]
\end{array}
$$

4.5 SUMMARY

In this chapter, a few important concepts and statements were expressed in relation-algebraic terms, which, using a proposed hylomorphic general form, allowed the derivation of the greedy algorithm for the color problem.

5

THINNING

This chapter develops the theory of the *thinning* operator, useful for the derivation of many Dynamic Programming algorithms. It covers the basic properties of thinning, along with laws for preorder thinning and recursive thinning.

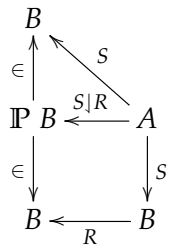
5.1 THINNING

From an algorithmic perspective, the mechanics of the *shrinking* operator are analogous to the greedy strategy – only a single result is collected at each stage, leaving no option to consider seemingly suboptimal choices. *Thinning* (Bird and de Moor, 1997) provides a way to consider multiple results at each stage of the algorithm, and a dynamic programming strategy can be implemented by excluding only those from which an optimal solution is impossible.

As suggested by Oliveira (2018), a notation similar to $S \upharpoonright R$ can be adopted for thinning,

$$S \lfloor R = \in \setminus S \cap (\in^\circ \cdot R) \,/\, S^\circ \tag{5.1}$$

where $\in \xleftarrow{B} \mathbb{P}\ B$ and $\mathbb{P}\ B \xleftarrow{S \lfloor R} A$ is a set-valued relation: $x\ (S \lfloor R)\ a$ holds for exactly those sets $x \subseteq \Lambda S\ a$ and x is lower-bounded with respect to R, as in the type diagram:



The universal property of $S \lfloor R$ arises by indirect equality:

$$X \subseteq S \lfloor R \ \equiv \ \begin{cases} \in \cdot X \subseteq S \\ X \cdot S^\circ \subseteq \in^\circ \cdot R \end{cases} \tag{5.2}$$

Clearly drawn from (5.1), $S \lfloor R$ is monotonic on the optimization criterion:

$$S \lfloor R \subseteq S \lfloor Q \;\Leftarrow\; R \subseteq Q \tag{5.3}$$

Note that $S \lceil R$ corresponds to that part of $S \lfloor R$ whose outputs are singletons containing minima, $\eta \cdot (S \lceil R) \;\subseteq\; S \lfloor R$ where $\eta\ b = \{b\}$.[1]

BASIC PROPERTIES OF THINNING This section proves useful (new or not proved in the literature) results about thinning. We start by the cancellation of the universal property, that is, by making $X := S \lfloor R$ in (5.2):

$$\in \cdot (S \lfloor R) \;\subseteq\; S \tag{5.4}$$
$$(S \lfloor R) \cdot S^\circ \;\subseteq\; \in^\circ \cdot R \tag{5.5}$$

These laws can also be expressed using relational divisions (2.91) and (2.108):

$$S \lfloor R \;\subseteq\; \in \backslash S \tag{5.6}$$
$$S \lfloor R \;\subseteq\; (\in^\circ \cdot R) \,/\, S^\circ \tag{5.7}$$

The first cancellation law tells us that thinning a relation only includes elements which already exist in the original relation. In other words, an element of a set resulting from thinning a relation S must be related to its corresponding input by S. The second tells that, wherever $x\ (S \lfloor R)\ a$ and $b\ S\ a$ hold, then some $b' \in x$ is such that $b'\ R\ b$.

Not surprisingly, shrinking and thinning share similar laws, cf. (Oliveira, 2018).

$$(S \cdot f) \lfloor R = (S \lfloor R) \cdot f \tag{5.8}$$
$$\frac{f}{g} \lfloor R = (\frac{id}{g} \lfloor R) \cdot f \tag{5.9}$$
$$S \lfloor R = S \lfloor (R \cap \mathrm{img}\ S) \tag{5.10}$$
$$S \lfloor id = \Lambda S \tag{5.11}$$
$$\bot \lfloor R = \Lambda \bot \tag{5.12}$$
$$[S, Q] \lfloor R = [S \lfloor R, Q \lfloor R] \tag{5.13}$$
$$S \lfloor R \subseteq (\in \cdot (S \lfloor R)) \lfloor R \tag{5.14}$$
$$S \lfloor R = (\in \lfloor R) \cdot \Lambda S \tag{5.15}$$

Proof of (5.8):

$$\begin{array}{ll} & X \subseteq (S \cdot f) \lfloor R \\ \equiv & \quad \{\ \text{universal property (5.2)}\ \} \end{array}$$

1 See (5.18) later on.

$$
\begin{array}{ll}
& \left\{ \begin{array}{l} \in \cdot X \subseteq S \cdot f \\ X \cdot (S \cdot f)^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\equiv & \quad \{ \text{ converse (2.4), shunting (2.59) } \} \\
& \left\{ \begin{array}{l} \in \cdot X \cdot f^\circ \subseteq S \\ X \cdot f^\circ \cdot S^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\equiv & \quad \{ \text{ universal property (5.2) } \} \\
& X \cdot f^\circ \subseteq S \mathbin{\backslash\!\!\rfloor} R \\
\equiv & \quad \{ \text{ shunting (2.59) } \} \\
& X \subseteq (S \mathbin{\backslash\!\!\rfloor} R) \cdot f \\
:: & \quad \{ \text{ indirect equality (2.9) } \} \\
& (S \cdot f) \mathbin{\backslash\!\!\rfloor} R = (S \mathbin{\backslash\!\!\rfloor} R) \cdot f \\
\Box &
\end{array}
$$

Proof of (5.9):

$$
\begin{array}{ll}
& \frac{f}{g} \mathbin{\backslash\!\!\rfloor} R \\
= & \quad \{ \text{ definition of metaphor (2.247) } \} \\
& (g^\circ \cdot f) \mathbin{\backslash\!\!\rfloor} R \\
= & \quad \{ \text{ thinning fusion (5.8) } \} \\
& (g^\circ \mathbin{\backslash\!\!\rfloor} R) \cdot f \\
= & \quad \{ \text{ definition of metaphor (2.247) with } f := id \text{ gives } \frac{id}{g} = g^\circ \ \} \\
& (\frac{id}{g} \mathbin{\backslash\!\!\rfloor} R) \cdot f \\
\Box &
\end{array}
$$

Proof of (5.10) by circular inclusion ("ping-pong"): $S \mathbin{\backslash\!\!\rfloor} (R \cap \text{img } S) \subseteq S \mathbin{\backslash\!\!\rfloor} R$ comes straight from (5.3), since $R \cap \text{img } S \subseteq R$. The "ping" part is not so immediate:

$$
\begin{array}{ll}
& S \mathbin{\backslash\!\!\rfloor} R \subseteq S \mathbin{\backslash\!\!\rfloor} (R \cap \text{img } S) \\
\equiv & \quad \{ \text{ universal property (5.2); cancellation (5.4) } \} \\
& (S \mathbin{\backslash\!\!\rfloor} R) \cdot S^\circ \subseteq \in^\circ \cdot (R \cap \text{img } S) \\
\equiv & \quad \{ \text{ unfolding definition (5.1) } \} \\
& (\in \setminus S \cap \in^\circ \cdot R \mathbin{/} S^\circ) \cdot S^\circ \subseteq \in^\circ \cdot (R \cap \text{img } S) \\
\Leftarrow & \quad \{ \text{ left semi-linearity of meet (2.23); cancellation (2.94) } \} \\
& (\in \setminus S) \cdot S^\circ \cap \in^\circ \cdot R \subseteq \in^\circ \cdot (R \cap \text{img } S)
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \{ \text{ modular law (2.33) } \} \\
& \in^\circ \cdot (\in \cdot (\in \setminus S) \cdot S^\circ \cap R) \subseteq \in^\circ \cdot (R \cap \mathrm{img}\ S) \\
\Leftarrow & \{ \text{ cancellation (2.93) } \} \\
& \in^\circ \cdot (S \cdot S^\circ \cap R) \subseteq \in^\circ \cdot (R \cap \mathrm{img}\ S) \\
\equiv & \{ \text{ trivia } \} \\
& \mathit{true}
\end{array}
$$

□

Proof of (5.11):

$$
\begin{array}{ll}
& S \lfloor id \\
= & \{ \text{ thinning definition (5.1) } \} \\
& \in \setminus S \cap \in^\circ / S^\circ \\
= & \{ \text{ symmetric division definition (2.110) } \} \\
& \dfrac{S}{\in} \\
= & \{ \text{ power transpose definition (2.135) } \} \\
& \Lambda S
\end{array}
$$

□

Proof of (5.12):

$$
\begin{array}{ll}
& \bot \lfloor R \\
= & \{ \text{ (5.1) ; (2.99) } \} \\
& \in \setminus \bot \\
= & \{ \text{ (2.144) } \} \\
& \Lambda \bot
\end{array}
$$

□

Proof of (5.13):

$$
\begin{array}{ll}
& X \subseteq [S, Q] \lfloor R \\
\equiv & \{ \text{ substitute } X := [X_1, X_2] \ \} \\
& [X_1, X_2] \subseteq [S, Q] \lfloor R \\
\equiv & \{ \text{ universal property of thinning (5.2) } \}
\end{array}
$$

$$
\begin{array}{ll}
& \left\{ \begin{array}{l} \in \cdot [X_1, X_2] \subseteq [S, Q] \\ [X_1, X_2] \cdot [S, Q]^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\equiv & \quad \{ \text{ coproduct fusion (2.83); biproduct absorption (2.85) } \} \\
& \left\{ \begin{array}{l} [\in \cdot X_1, \in \cdot X_2] \subseteq [S, Q] \\ X_1 \cdot S^\circ \cup X_2 \cdot Q^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\equiv & \quad \{ \text{ coproduct inclusion (2.81); universal property of join (2.14) } \} \\
& \left\{ \begin{array}{l} \in \cdot X_1 \subseteq S \\ \in \cdot X_2 \subseteq Q \\ X_1 \cdot S^\circ \subseteq \in^\circ \cdot R \\ X_2 \cdot Q^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\equiv & \quad \{ \text{ universal property of thinning (5.2) twice } \} \\
& \left\{ \begin{array}{l} X_1 \subseteq S \lfloor R \\ X_2 \subseteq Q \lfloor R \end{array} \right. \\
\equiv & \quad \{ \text{ coproduct inclusion (2.81); } X := [X_1, X_2] \ \} \\
& X \subseteq [S \lfloor R, Q \lfloor R] \\
:: & \quad \{ \text{ indirect equality (2.9) } \} \\
& [S, Q] \lfloor R = [S \lfloor R, Q \lfloor R]
\end{array}
$$

□

Proof of (5.14):

$$
\begin{array}{ll}
& S \lfloor R \ \subseteq \ (\in \cdot (S \lfloor R)) \lfloor R \\
\equiv & \quad \{ \text{ universal property (5.2) } \} \\
& \left\{ \begin{array}{l} \in \cdot (S \lfloor R) \ \subseteq \ \in \cdot (S \lfloor R) \\ (S \lfloor R) \cdot (\in \cdot (S \lfloor R))^\circ \ \subseteq \ \in^\circ \cdot R \end{array} \right. \\
\Leftarrow & \quad \{ \text{ thin cancellation (5.4) } \} \\
& (S \lfloor R) \cdot S^\circ \ \subseteq \ \in^\circ \cdot R \\
\Leftarrow & \quad \{ \text{ thin cancellation (5.5) } \} \\
& true
\end{array}
$$

□

Proof of 5.15:

$$
\begin{array}{ll}
& (\in \lfloor R) \cdot \Lambda S \\
= & \quad \{ \text{ thinning fusion (5.8) } \}
\end{array}
$$

$$
\begin{array}{ll}
& \in \cdot \Lambda S \mathbin{\rfloor} R \\
= & \quad \{ \text{ power transpose cancellation (2.139) } \} \\
& S \mathbin{\rfloor} R \\
\Box &
\end{array}
$$

Thinning $\in \mathbin{\rfloor} R$ distributes over union (see proof in Appendix B),

$$\mu \cdot \mathbb{P}\ (\in \mathbin{\rfloor} R) \subseteq (\in \mathbin{\rfloor} R) \cdot \mu \tag{5.16}$$

which leads to the following partitioning rule:

$$\mu \cdot \mathbb{P}\ (S \mathbin{\rfloor} R) \cdot \Lambda T \subseteq S \cdot T \mathbin{\rfloor} R \tag{5.17}$$

Proof:

$$
\begin{array}{ll}
& S \cdot T \mathbin{\rfloor} R \\
= & \quad \{ \text{ (5.15) } \} \\
& (\in \mathbin{\rfloor} R) \cdot \Lambda(S \cdot T) \\
= & \quad \{ \text{ transpose of composition (2.232); Kleisli composition (2.227) } \} \\
& (\in \mathbin{\rfloor} R) \cdot \mu \cdot \mathbb{P}\ \Lambda S \cdot \Lambda T \\
\supseteq & \quad \{ \text{ (5.16) } \} \\
& \mu \cdot \mathbb{P}\ (\in \mathbin{\rfloor} R) \cdot \mathbb{P}\ \Lambda S \cdot \Lambda T \\
= & \quad \{ \text{ power relator (2.150); (5.15) } \} \\
& \mu \cdot \mathbb{P}\ (S \mathbin{\rfloor} R) \cdot \Lambda T \\
\Box &
\end{array}
$$

Another step from thinning to shrinking is the following *thin-elimination* refinement step:

$$\eta \cdot (S \mathbin{\restriction} R) \subseteq S \mathbin{\rfloor} Q \quad \Leftarrow \quad R \cap \mathrm{img}\ S \subseteq Q \tag{5.18}$$

Proof of (5.18):

$$
\begin{array}{ll}
& \eta \cdot (S \mathbin{\restriction} R) \subseteq S \mathbin{\rfloor} Q \\
\equiv & \quad \{ \text{ universal property (5.2) } \} \\
& \left\{ \begin{array}{l} \in \cdot \eta \cdot (S \mathbin{\restriction} R) \subseteq S \\ \eta \cdot (S \mathbin{\restriction} R) \cdot S^\circ \subseteq \in^\circ \cdot Q \end{array} \right. \\
\equiv & \quad \{ \ \in \cdot \eta = id \text{; shrink cancellation (2.187); shunting (2.58); converse (2.4) } \} \\
& (S \mathbin{\restriction} R) \cdot S^\circ \subseteq (\in \cdot \eta)^\circ \cdot Q
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \{\ \in \cdot \eta = id, R \cap \mathrm{img}\ S \subseteq Q\ \} \\
& (S \upharpoonright R) \cdot S^{\circ} \subseteq R \cap \mathrm{img}\ S \\
\equiv & \{\ \text{shrinking cancellation (2.188)}\ \} \\
& R \subseteq R \cap \mathrm{img}\ S \\
\equiv & \{\ \text{universal property (2.13)}\ \} \\
& true
\end{array}
$$

□

5.2 PREORDER THINNING

For R a preorder, which is generally the case in optimization problems, a number of laws of the thinning operator come in handy. All laws laid out in the rest of this chapter deal with preorders, unless a weaker condition is specified (e.g. "R is reflexive").

The least such relation is the identity ($R = id$). When the thinning relation is the identity, there is no restriction on the outputs to select and thus $S \lfloor id = \Lambda S$, recall (5.11). From (5.11) and thinning-monotonicity (5.3) one draws

$$\Lambda S \subseteq S \lfloor R \ \Leftarrow\ R \text{ is reflexive} \tag{5.19}$$

Thus, for reflexive R, $S \lfloor R$ is always larger than function ΛS. Since larger than entire is entire (2.48), we conclude that $S \lfloor R$ is always entire for reflexive R.

From (5.19) we get $S \subseteq \in \cdot (S \lfloor R)$ and therefore:

$$\in \cdot (S \lfloor R) = S \ \Leftarrow\ R \text{ reflexive} \tag{5.20}$$

For reflexive thinning relations, the identity always thins to "return" (in the powerset monad):

$$id \lfloor R = \eta \ \Leftarrow\ R \text{ is reflexive} \tag{5.21}$$

Proof: $\eta \subseteq id \lfloor R$ is immediate because R is reflexive and thus $\eta = \Lambda id \subseteq id \lfloor R$ by (5.19). The proof for the ping step $id \lfloor R \subseteq \eta$ is less immediate:

$$
\begin{array}{ll}
& id \lfloor R \subseteq \eta \\
\equiv & \{\ \text{thinning definition (5.1)}; \eta = \frac{id}{\in}\ \text{etc}\ \} \\
& \in \setminus id \cap \in^{\circ} \cdot R \subseteq \in \setminus id \cap \in^{\circ} \\
\equiv & \{\ \text{thinning definition (5.1)}; \eta = \frac{id}{\in}\ \text{etc}\ \} \\
& \in \setminus id \cap \in^{\circ} \cdot R \subseteq \in^{\circ} \\
\Leftarrow & \{\ \text{ignore } R \text{ (raise the lower side) (2.32)}\ \}
\end{array}
$$

$$
\begin{array}{ll}
& \in \setminus id \cap \in^\circ \cdot \top \subseteq \in^\circ \\
\Leftarrow & \quad \{ \text{ modular law (2.34) } \} \\
& \in^\circ \cdot (\in \cdot (\in \setminus id) \cap \top) \subseteq \in^\circ \\
\Leftarrow & \quad \{ \text{ } \top \text{ above everything (2.51); division cancellation (2.93) } \} \\
& \in^\circ \cdot id \subseteq \in^\circ \\
\equiv & \quad \{ \text{ trivial } \} \\
& true \\
\Box &
\end{array}
$$

Then, by thinning fusion (5.8), thinning a function is "returning it" (in the powerset monad):

$$f \lfloor R = \eta \cdot f \quad \Leftarrow \quad R \text{ is reflexive} \tag{5.22}$$

Proof:

$$
\begin{array}{ll}
& f \lfloor R \\
= & \quad \{ \text{ thinning fusion (5.8) } \} \\
& (id \lfloor R) \cdot f \\
= & \quad \{ \text{ identity thinning (5.21) } \} \\
& \eta \cdot f \\
\Box &
\end{array}
$$

For preorders, shrinking can be expressed in terms of thinning:

$$S \lceil R = (\in \lceil R) \cdot (S \lfloor R) \tag{5.23}$$

This is useful to convert a *shrinking problem* into a *thinning problem*. Indeed, many problems specified by shrinking have a dynamic programming solution calculated via thinning. The proof of (5.23) is discharged by ping-pong. The ping step makes use of the power transpose:

$$
\begin{array}{ll}
& S \lceil R \subseteq (\in \lceil R) \cdot (S \lfloor R) \\
\Leftarrow & \quad \{ \text{ } R \text{ is reflexive; monotonicity (2.31) } \} \\
& S \lceil R \subseteq (\in \lceil R) \cdot (S \lfloor id) \\
\equiv & \quad \{ \text{ thinning by identity (5.11) } \} \\
& S \lceil R \subseteq (\in \lceil R) \cdot \Lambda S \\
\equiv & \quad \{ \text{ shrinking fusion with power transpose by (5.19) and (5.23) } \} \\
& S \lceil R \subseteq S \lceil R \\
\Box &
\end{array}
$$

The pong step is a simple application of the universal property and cancellation laws of thinning and shrinking:

$$
\begin{array}{ll}
 & (\in \upharpoonright R) \cdot (S \downharpoonright R) \subseteq S \upharpoonright R \\
\equiv & \quad \{ \text{ universal property (5.2) } \} \\
 & \left\{ \begin{array}{l} (\in \upharpoonright R) \cdot (S \downharpoonright R) \subseteq S \\ (\in \upharpoonright R) \cdot (S \downharpoonright R) \cdot S^{\circ} \subseteq R \end{array} \right. \\
\Leftarrow & \quad \{ \text{ shrinking and thinning cancellation (2.187,5.5) } \} \\
 & \left\{ \begin{array}{l} \in \cdot (S \downharpoonright R) \subseteq S \\ (\in \upharpoonright R) \cdot \in^{\circ} \cdot R \subseteq R \end{array} \right. \\
\Leftarrow & \quad \{ \text{ shrinking and thinning cancellation (2.188,5.4) } \} \\
 & R \cdot R \subseteq R \\
\equiv & \quad \{ \text{ } R \text{ is assumed transitive (2.37) } \} \\
 & true \\
\Box &
\end{array}
$$

The proof above is easily extended to

$$S \upharpoonright R = (\in \upharpoonright R) \cdot (S \downharpoonright Q) \;\Leftarrow\; Q \subseteq R \tag{5.24}$$

which itself extends the rule called *thin-introduction* in (Bird and de Moor, 1997). As mentioned before, shrinking the membership, $\in \upharpoonright R$, is what Bird and de Moor (1997) call $min\ R$. The corresponding for thinning, $\in \downharpoonright R$ is termed $thin\ R$. By cancellation (5.6), $thin\ R$ is a subrelation of $\in \setminus \in$, i.e. it is smaller than set inclusion. By (5.8) and (2.139):

$$S \downharpoonright R = (\in \downharpoonright R) \cdot \Lambda S \tag{5.25}$$

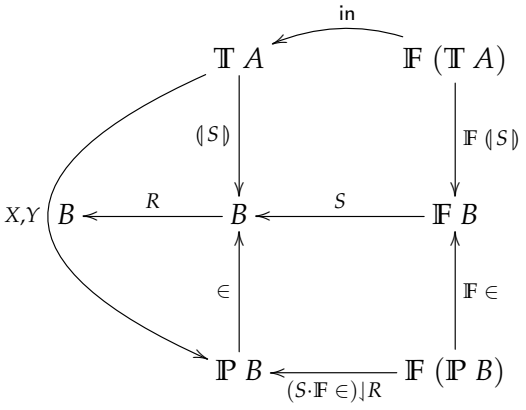
5.3 RECURSIVE THINNING

The following theorem, first presented by Bird and de Moor (1997), will allow us to promote thinning into a catamorphism by showing that it is possible, subject to conditions, to perform the thinning step in each recursive invocation, instead of at the end. Here we present the full proof, using the thinning combinator proposed by Oliveira (2018).

Theorem 5.1. *For preorder R and S $\mathbb{F}$-monotonic with respect to R°, we have:*

$$(\!|\, (S \cdot \mathbb{F} \in) \downharpoonright R \,|\!) \subseteq (\!|\, S \,|\!) \downharpoonright R \tag{5.26}$$

Diagram, where X abbreviates $(\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!)$ *and Y abbreviates* $(\!|\, S \,|\!) \rfloor\ R$



Proof.

$$
\begin{array}{ll}
& (\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \subseteq (\!|\, S \,|\!) \rfloor\ R \\
\equiv & \quad \{\ \textit{universal property (5.2)}\ \} \\
& \left\{ \begin{array}{l} \in \cdot\ (\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \subseteq (\!|\, S \,|\!) \\ (\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \cdot (\!|\, S \,|\!)^\circ \subseteq \in^\circ \cdot R \end{array} \right.
\end{array}
$$

Handling $\in \cdot\ (\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \subseteq (\!|\, S \,|\!)$*:*

$$
\begin{array}{ll}
& \in \cdot\ (\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \subseteq (\!|\, S \,|\!) \\
\Leftarrow & \quad \{\ \textit{cata-fusion (2.213)}\ \} \\
& \in \cdot\ ((S \cdot \mathbb{F}\ \in) \rfloor\ R) \subseteq S \cdot \mathbb{F}\ \in \\
\Leftarrow & \quad \{\ \textit{thin cancellation (5.4)}\ \} \\
& S \cdot \mathbb{F}\ \in \subseteq S \cdot \mathbb{F}\ \in \\
& \square
\end{array}
$$

Handling $(\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \cdot (\!|\, S \,|\!)^\circ \subseteq \in^\circ \cdot R$*:*

$$
\begin{array}{ll}
& (\!|\, (S \cdot \mathbb{F}\ \in) \rfloor\ R \,|\!) \cdot (\!|\, S \,|\!)^\circ \subseteq \in^\circ \cdot R \\
\Leftarrow & \quad \{\ \textit{hylomorphism least fixpoint (2.220)}\ \} \\
& ((S \cdot \mathbb{F}\ \in) \rfloor\ R) \cdot \mathbb{F}\ (\in^\circ \cdot R) \cdot S^\circ \subseteq \in^\circ \cdot R \\
\equiv & \quad \{\ \textit{functors (2.130)}\ \} \\
& ((S \cdot \mathbb{F}\ \in) \rfloor\ R) \cdot \mathbb{F}\ \in^\circ \cdot \mathbb{F}\ R \cdot S^\circ \subseteq \in^\circ \cdot R \\
\Leftarrow & \quad \{\ \textit{assumption: } S \textit{ is monotonic on } R^\circ\textit{, (2.132) applies}\ \} \\
& ((S \cdot \mathbb{F}\ \in) \rfloor\ R) \cdot \mathbb{F}\ \in^\circ \cdot S^\circ \cdot R \subseteq \in^\circ \cdot R \\
\equiv & \quad \{\ \textit{converse (2.4)}\ \}
\end{array}
$$

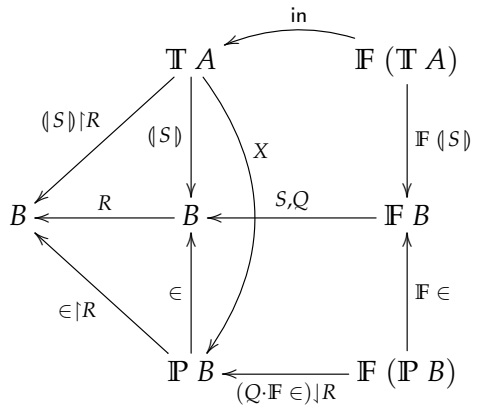
$$
\begin{array}{ll}
 & ((S \cdot \mathbb{F}\in) \lfloor R) \cdot (S \cdot \mathbb{F}\in)^\circ \cdot R \subseteq \in^\circ \cdot R \\
\Leftarrow & \quad \{\ \textit{thin cancellation (5.5)}\ \} \\
 & \in^\circ \cdot R \cdot R \subseteq \in^\circ \cdot R \\
\Leftarrow & \quad \{\ \textit{R is assumed transitive}\ \} \\
 & \in^\circ \cdot R \subseteq \in^\circ \cdot R \\
 & \square
\end{array}
$$

□

Corollary 5.1. *For a preorder R, $Q \subseteq R$ and assuming S $\mathbb{F}$-monotonic with respect to Q°, we have:*

$$(\in \upharpoonright R) \cdot (\!|\, (S \cdot \mathbb{F}\in) \lfloor Q \,|\!) \subseteq (\!|\, S \,|\!) \upharpoonright R \tag{5.27}$$

Diagram, where X abbreviates $(\!|\, (S \cdot \mathbb{F}\in) \lfloor Q \,|\!)$:



Proof:

$$
\begin{array}{ll}
 & (\in \upharpoonright R) \cdot (\!|\, (S \cdot \mathbb{F}\in) \lfloor Q \,|\!) \\
\subseteq & \quad \{\ \textit{extract thinning from catamorphism (5.26)}\ \} \\
 & (\in \upharpoonright R) \cdot ((\!|\, S \,|\!) \lfloor Q) \\
\subseteq & \quad \{\ \textit{monotonicity of thinning criterion (5.3), } Q \subseteq R \textit{ assumed}\ \} \\
 & (\in \upharpoonright R) \cdot ((\!|\, S \,|\!) \lfloor R) \\
= & \quad \{\ \textit{thinning introduction (5.23)}\ \} \\
 & (\!|\, S \,|\!) \upharpoonright R \\
 & \square
\end{array}
$$

In the case of a shrinking hylomorphism, optimization is possible if we can avoid constructing a complete solution in each recursive call. If we can find a measure that allows us to conclude, at the intermediate stage,

that the candidate solution to be generated will not be the optimal one, then we can exclude it outright and avoid actually generating it.

Theorem 5.2. *Let* $H = (\!|\, h \,|\!) \cdot (\!|\, T \,|\!)^\circ$ *and* $M = H \upharpoonright R$. *We have:*

$$\langle \mu\, X \;::\; (h \cdot \in \upharpoonright R) \cdot \mathbb{P}\, \mathbb{F}\, X \cdot (T^\circ \lfloor\, Q) \rangle \;\subseteq\; M \tag{5.28}$$

provided h *is monotonic with respect to* R *and* $h \cdot \mathbb{F}\, H \cdot Q^\circ \;\subseteq\; R^\circ \cdot h \cdot \mathbb{F}\, H$.
Proof: see Appendix B.

□

5.4 IMPLEMENTING THINNING

This section provides a theoretical framing for an efficient implementation of thinning, which is taken from (Mu et al., 2010). Relations involving thinning are more often than not nondeterministic. So, in general, there are many possible implementations of thinning. One may wish to obtain a function from the general specification

$$\in \lfloor\, Q \;\supseteq\; thin\, Q \tag{5.29}$$

but, for the purposes of this work, it will suffice to take $Q := R \cap S$, for$R = (\geqslant)_f$ and a preorder S, and assume the input is (descendingly) ordered with respect to R, such that:

$$(\in \lfloor\, (R \cap S)) \cdot \phi_{orderedBy\ R} \;\supseteq\; bump\, f\ S \tag{5.30}$$

The $bump$ function is linear time algorithm which sequentially processes adjacent pairs. For each pair of elements x and y, it determines whether either can be excluded from the result.

Given the specification, we can assume $f\ x \geqslant f\ y$, and so we are left with three main cases:

- when $x\ S\ y$ holds, $x\ Q\ y$ also holds and y can be excluded;
- otherwise, if $f\ x = f\ y$, $y\ Q\ s$ holds and x can be excluded;
- otherwise, $f\ x > f\ y$ but $y\ S\ x$, so neither $x\ Q\ y$ nor $y\ Q\ x$ hold, and neither is excluded.

When y is excluded, the algorithm proceeds by comparing x and the element succeeding y. The algorithm ends when the list is empty or has a single element.

$$\begin{aligned}
&bump\, f\ g\ [\,] = [\,] \\
&bump\, f\ g\ [x] = [x] \\
&bump\, f\ g\ (x : y : xs) \\
&\quad \mid x\ `g`\ y = bump\, f\ g\ (x : xs) \\
&\quad \mid vx \equiv vy = bump\, f\ g\ (y : xs) \\
&\quad \mid vx > vy = x : bump\, f\ g\ (y : xs) \\
&\textbf{where}\ (vx, vy) = (f\ x, f\ y)
\end{aligned}$$

5.5 SUMMARY

In this chapter, the theory underlying the *thinning* operator was expanded, showing that thinning and shrinking share many of the basic properties. Thinning introduction and elimination laws interacting with shrinking, preorder thinning, and recursive thinning law were also covered.

6

DYNAMIC PROGRAMMING

This chapter describes Dynamic Programming (DP) problems and proposes an algorithmic characterization of the problem space. The role of thinning in DP is discussed, and its application demonstrated through two case studies.

We begin by delving into important concepts in Dynamic Programming, characterizing the problem space by the types of algorithms used to solve DP problems, and giving some examples taken from the problem catalogue given in Appendix A. Recall from chapter 1 that DP problems are characterized by two essential properties: optimal substructure and overlapping subproblems. A problem with an optimal structure, also said to follow the 'Principle of Optimality', is one in which the optimal policy for the current decision rests on the existence of an optimal policy for all future decisions. Overlapping subproblems are problems of smaller size which are necessary to solve multiple larger problems. So, ideally, they should not be recalculated.

Problems exhibiting optimal substructure but any without overlapping subproblems are not solved by DP techniques. Dynamic Programming necessarily implies the use of optimal substructure to avoid repeated calculations of overlapping subproblems. Instead, a divide and conquer approach is taken, which leads to an efficient and parallelizable algorithm.

It is more productive to refer to algorithms rather than problems when characterizing Dynamic Programming. This is because some problems, although they fit squarely in the DP problem space, may be solved by different types of algorithms. For instance, calculating Fibonacci numbers is a classic DP problem, but one particular implementation, which uses the tupling strategy to maintain only the previous two numbers in the sequence, is more akin to a greedy algorithm than a DP one.

Although this tupling strategy may seem like a variation of a tabulation method, the two are fundamentally different. The main difference is what is known before runtime in each case. Where only a tabulation method is possible, the resulting table contains entries which may or may not be useful in calculating the final result. Moreover, the order in which they will be used is only determined at runtime. In the tupling strategy, it has been statically determined that only a constant number of memory cells are needed, because there is an underlying order by which solutions are computed. This algorithm resembles the greedy algorithm, only changing how many solutions are calculated at each step of the process.

THINNING The concept of thinning is very important to Dynamic Programming. Most problems do not require searching over the whole solution space, because they possess properties which allow its culling. Thinning

does just this. It is the technique utilizing these properties to exclude solutions which are not useful for calculating the optimal one.

Not every DP algorithm makes use of thinning. Some problems with optimal substructure and overlapping subproblems still require a full search of the solution space. This search is optimized by trading space for time, most commonly through a tabulation scheme. An example of this is the problem of building minimum cost trees, formulated as a specific instance of the "optimal bracketing problem" (section A.15).

Other algorithms may use thinning as a starting point, but, in the end, operate by a logic of partitioning the problem into multiple maximization problems. The "paths in a layered network" (A.10) and "shortest paths on a cylinder" (A.7) problems illustrate this strategy, as only a single path per vertex (in the current set) is considered, and at the end the best one is chosen. The difference between this so-called 'partitioning' strategy and a divide and conquer algorithm, is that the generated solutions are complete, and so only one will be chosen, while in the latter each division is a solution to only part of the problem, and needs to be combined with the others to obtain the result.

Finally, we single out two different but related ways to do thinning: 'decomposition' thinning and 'explicit' thinning. Decomposition thinning consists in restricting the ways in which new solutions are generated (in other words, of choosing only useful decompositions). This type of method is common in problems specified by the converse of a function, for example, the "string edit problem" (A.14). Explicit thinning, on the other hand, uses a modified criterion to compare all generated solutions, and at the end of each step, exclude those that are not useful. The "knapsack problem" (A.3) is the classic example of this type of algorithm. In general, it is preferable to act earlier and exclude decompositions from consideration, avoiding generating more solutions and having to compare them to others to determine their usefulness.

6.1 KNAPSACK

The 0-1 knapsack problem is a good example of a combinatorial optimization problem. This section presents a solution to the knapsack problem as laid out by Bird and de Moor (1997), followed by a generalized specification and derivation method for the problem.

> Consider a set of N items, two functions, respectively named wt and val, that give the weight and the value of a given item, and a weight limit $w > 0$. The value/weight of a set of items is the sum of the value/weight of each of its items. Determine the subset of items with the highest value, while still abiding by the weight limit, that is, its weight should be no greater than w.

Representing sets of items as sequences, the problem is specified as the selection of greatest valued subsequence, post-conditioning the generating relation by the appropriate predicate $within\ w$:

$$
\begin{aligned}
&Knapsack = \phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value} \ \mathbf{where} \\
&\quad (\sqsubseteq) = (\!|\, [nil, cons \cup \pi_2] \,|\!) \\
&\quad value = sum \cdot val^*
\end{aligned}
$$

$$within\ w\ x = weight\ x\ (\leqslant)\ w$$
$$weight = sum \cdot wt^*$$

The set system underlying the knapsack problem is hereditary, as the empty list is below capacity, and every sublist has lower weight than the original. This means that both (4.2) and (4.5) hold for $p := within\ w$.

Applying the thinning theorem (5.26) will exclude candidate solutions that are strictly worse than others. The optimizing relation used needs to be stronger than $(\geqslant)_{value}$, so

$$(\preccurlyeq) := (\geqslant)_{value} \cap (\leqslant)_{weight} \tag{6.1}$$

meets the requirement. For the main derivation, with $p := within\ w$, we have:

$$\begin{aligned}
& \phi_p \cdot (\!|\, [nil, cons \cup \pi_2] \,|\!) \upharpoonright (\geqslant)_{value} \\
= & \quad \{ \text{ cata-fusion (2.213) — details below } \} \\
& (\!|\, [nil, \phi_p \cdot cons \cup \pi_2] \,|\!) \upharpoonright (\geqslant)_{value} \\
\supseteq & \quad \{ \text{ thin-introduction (5.24); thinning theorem (5.26) } \} \\
& (\in \upharpoonright (\geqslant)_{value}) \cdot (\!|\, [nil, \phi_p \cdot cons \cup \pi_2] \cdot \mathbb{F} \in \downharpoonright (\preccurlyeq) \,|\!) \\
= & \quad \{ \ \mathbb{F}\ R = id + id \times R \text{ for cons lists, direct sum absorption (2.87) } \} \\
& (\in \upharpoonright (\geqslant)_{value}) \cdot (\!|\, [nil, (\phi_p \cdot cons \cup \pi_2) \cdot (id \times \in)] \downharpoonright (\preccurlyeq) \,|\!) \\
= & \quad \{ \text{ bilinearity of composition with join (2.21) } \} \\
& (\in \upharpoonright (\geqslant)_{value}) \cdot (\!|\, [nil, \phi_p \cdot cons \cdot (id \times \in) \cup \pi_2 \cdot (id \times \in)] \downharpoonright (\preccurlyeq) \,|\!) \\
= & \quad \{ \text{ product cancellation (2.73) } \} \\
& (\in \upharpoonright (\geqslant)_{value}) \cdot (\!|\, [nil, \phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2] \downharpoonright (\preccurlyeq) \,|\!) \\
= & \quad \{ \text{ thinning coproduct (5.13); function thinning (5.22) } \} \\
& (\in \upharpoonright (\geqslant)_{value}) \cdot (\!|\, [\eta \cdot nil, (\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2) \downharpoonright (\preccurlyeq)] \,|\!)
\end{aligned}$$

The fusion step involves finding a relation S satisfying

$$\phi_p \cdot [nil, cons \cup \pi_2] = S \cdot \mathbb{F}\ \phi_p$$

which can be derived by factoring the left-hand side:

$$\begin{aligned}
& \phi_p \cdot [nil, cons \cup \pi_2] \\
= & \quad \{ \text{ coproduct fusion (2.83) } \} \\
& [\phi_p \cdot nil, \phi_p \cdot (cons \cup \pi_2)] \\
= & \quad \{ \text{ trivial axiom (4.2); right linearity of join (2.20) } \} \\
& [nil, \phi_p \cdot cons \cup \phi_p \cdot \pi_2] \\
= & \quad \{ \text{ (4.5); Kronecker product cancellation (2.75) } \}
\end{aligned}$$

$$
\begin{array}{ll}
& [nil, \phi_p \cdot cons \cdot (id \times \phi_p) \cup \pi_2 \cdot (id \times \phi_p)] \\
= & \quad \{ \text{ left linearity of join (2.21) } \} \\
& [nil, (\phi_p \cdot cons \cup \pi_2) \cdot (id \times \phi_p)] \\
= & \quad \{ \text{ direct sum absorption (2.87) } \} \\
& \underbrace{[nil, \phi_p \cdot cons \cup \pi_2]}_{S} \cdot \mathbb{F}\ \phi_p
\end{array}
$$

Next, we refine $(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2) \lfloor (\preccurlyeq)$ to a set-valued function:

$$
\begin{array}{ll}
& (\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2) \lfloor (\preccurlyeq) \\
= & \quad \{ \text{ (5.25) } \} \\
& (\in \lfloor (\preccurlyeq)) \cdot \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2) \\
\supseteq & \quad \{ \text{ refine using } \phi_{orderedBy\ R} \subseteq id \text{ (2.160); (5.30) } \} \\
& bump\ value\ ((\leqslant)_{weight}) \cdot \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2)
\end{array}
$$

The resulting equation for the *Knapsack* relation

$$Knapsack = (\in \upharpoonright (\geqslant)_{value}) \cdot (\!| [\eta \cdot nil, bump\ value\ ((\leqslant)_{weight}) \cdot \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2)] |\!)$$

can be implemented as a function over the powerset monad, using the non-determinism to manage the set of solutions at each point of the algorithm.

$$
\begin{array}{ll}
& \Lambda Knapsack \\
= & \quad \{ \text{ definition of } Knapsack \ \} \\
& \Lambda((\in \upharpoonright (\geqslant)_{value}) \cdot (\!| [\eta \cdot nil, bump\ value\ ((\leqslant)_{weight}) \cdot \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2)] |\!)) \\
= & \quad \{ \ \Lambda(R \cdot f) = \Lambda R \cdot f \text{ (2.141) } \} \\
& \Lambda(\in \upharpoonright (\geqslant)_{value}) \cdot (\!| [\eta \cdot nil, bump\ value\ ((\leqslant)_{weight}) \cdot \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2)] |\!) \\
\supseteq & \quad \{ \ shrink\ R \subseteq \Lambda(\in \upharpoonright R) \text{ (2.263) } \} \\
& (shrink\ (\geqslant)_{value}) \cdot (\!| [\eta \cdot nil, bump\ value\ ((\leqslant)_{weight}) \cdot \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2)] |\!)
\end{array}
$$

Finally, we calculate the functional implementation of the transpose by introducing variable arguments:

$$
\begin{array}{ll}
& \Lambda(\phi_p \cdot cons \cdot (id \times \in) \cup \in \cdot \pi_2)\ (a, xs) \\
= & \quad \{ \text{ power-transpose of join (2.236) } \} \\
& (\Lambda(\phi_p \cdot cons \cdot (id \times \in))\ (a, xs)) \oplus (\Lambda(\in \cdot \pi_2)\ (a, xs)) \\
= & \quad \{ \text{ power-transpose cancellation (2.142); } \pi_2\ (x, y) = y \ \} \\
& (\Lambda(\phi_p \cdot cons \cdot (id \times \in))\ (a, xs)) \oplus xs \\
= & \quad \{ \text{ power-transpose of relation composition (2.232) } \}
\end{array}
$$

$$
\begin{aligned}
& ((\Lambda(\phi_p \cdot cons) \bullet \Lambda(id \times \in))\ (a, xs)) \oplus xs \\
= & \quad \{\ \text{introduce } \mathbf{do} \text{ notation (2.228)}\ \} \\
& (\mathbf{do}\ \{(b, x) \leftarrow \Lambda(id \times \in)\ (a, xs); \Lambda(\phi_p \cdot cons)\ (b, x)\}) \oplus xs \\
= & \quad \{\ \text{product transpose (2.240); one-point rule (2.241)}\ \} \\
& (\mathbf{do}\ \{x \leftarrow xs; \Lambda(\phi_p \cdot cons)\ (a, x)\}) \oplus xs \\
= & \quad \{\ \Lambda(R \cdot f) = \Lambda R \cdot f \text{ (2.141); power transpose of coreflexive (2.243)}\ \} \\
& (\mathbf{do}\ \{x \leftarrow xs; guard\ (within\ w)\ (a : x)\}) \oplus xs
\end{aligned}
$$

We now have a function *knapsack*, which we directly convert into Haskell code:

$$
\begin{aligned}
& guard\ r = r \to return, nil \\
& within\ w\ x = weight\ x \leqslant w \\
& r\ a\ b = value\ a \geqslant value\ b \\
& q\ a\ b = weight\ a \leqslant weight\ b \\
& knapsack\ w = shrink\ r \cdot (\!|\ [return \cdot nil, step]\ |\!)\ \mathbf{where} \\
& \quad step\ (a, xs) = bump\ value\ q\ (newK\ (a, xs) \oplus xs) \\
& \quad newK\ (a, xs) = \mathbf{do}\ \{x \leftarrow xs; guard\ (within\ w)\ (a : x)\}
\end{aligned}
$$

The correctness of the algorithm is guaranteed by making the internal representation of powerset monad a list with the solutions sorted by the descending order of their value (as required by (5.30), otherwise the whole relation bottoms out). Morihata et al. (2014) achieve this result through incrementalization by making $\oplus = merge$ the union operation on lists of candidates.

GENERALIZED DERIVATION Substituting $(\trianglelefteq) = (\sqsubseteq) \cdot \frac{bag}{bag}$ for $(\sqsubseteq)$ in the first specification, we obtain a generalized one, which considers all feasible solutions from the start.

$$Knapsack = \phi_{within\ w} \cdot (\trianglelefteq) \upharpoonright (\geqslant)_{value}$$

The rest of this section shows how to get a specification in the form of the original one from the generalized one. This corresponds to proving that disregarding permutations will still yield an optimal solution. In relational algebra terms, we reason:

$$
\begin{aligned}
& \phi_{within\ w} \cdot (\trianglelefteq) \upharpoonright (\geqslant)_{value} \\
= & \quad \{\ \text{definition of } (\trianglelefteq) \text{ (3.43)}\ \} \\
& \phi_{within\ w} \cdot (\sqsubseteq) \cdot Perm \upharpoonright (\geqslant)_{value} \\
\supseteq & \quad \{\ \text{shrinking monotonicity (2.195)}\ \} \\
& \phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value}
\end{aligned}
$$

To justify the use of (2.195), we must prove the following two conditions:

$$\begin{cases} \phi_{within\ w} \cdot (\sqsubseteq) \subseteq \phi_{within\ w} \cdot Perm \cdot (\sqsubseteq) \\ (\phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value}) \cdot (\phi_{within\ w} \cdot Perm \cdot (\sqsubseteq))^\circ \subseteq (\geqslant)_{value} \end{cases}$$

Since $Perm = \frac{bag}{bag}$ is reflexive, the first condition is easily satisfied. The second asks us to prove that an optimal solution is reachable with the reduced number of candidate solutions.

$$\begin{array}{ll} & (\phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value}) \cdot (\phi_{within\ w} \cdot (\sqsubseteq) \cdot Perm)^\circ \subseteq (\geqslant)_{value} \\ \equiv & \quad \{\ (\sqsubseteq) \cdot Perm = Perm \cdot (\sqsubseteq)\ (3.47)\ \} \\ & (\phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value}) \cdot (\phi_{within\ w} \cdot Perm \cdot (\sqsubseteq))^\circ \subseteq (\geqslant)_{value} \\ \equiv & \quad \{\ within\ w \text{ is stable, so } \phi_{within\ w} \cdot Perm = Perm \cdot \phi_{within\ w}\ (3.34)\ \} \\ & (\phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value}) \cdot (Perm \cdot \phi_{within\ w} \cdot (\sqsubseteq))^\circ \subseteq (\geqslant)_{value} \\ \equiv & \quad \{\ \text{converse}\ \} \\ & (\phi_{within\ w} \cdot (\sqsubseteq) \upharpoonright (\geqslant)_{value}) \cdot (\phi_{within\ w} \cdot (\sqsubseteq))^\circ \cdot Perm \subseteq (\geqslant)_{value} \\ \Leftarrow & \quad \{\ \text{shrinking cancellation (2.188)}\ \} \\ & (\geqslant)_{value} \cdot Perm \subseteq (\geqslant)_{value} \\ \equiv & \quad \{\ value \text{ is stable, so } R_{value} \cdot Perm = R_{value}\ (3.33)\ \} \\ & true \end{array}$$

□

With this, we have proven that the commonly used specification is a valid specialization of the one provided here. It relies on the fact that neither the objective function nor the criterion of feasibility depend on the ordering of the elements in the list. But, as seen in the previous section, it is not always fruitful to disregard this ordering, so having a more general specification may help solve other problems with similar structure.

6.2 GENERAL LAYERED NETWORKS

It is possible to generalize the "paths in a layered network" problem tackled by Bird and de Moor (1997) and included in Appendix A. The original problem has the following description:

> A layered network is a non-empty sequence of sets of vertices. A path in a layered network is a sequence of vertices constructed by choosing a single vertex from each set, and its cost is the sum of the weight of each transition between adjacent vertices. Determine the path of least cost.

To generalize this problem, a layered network, instead of a non-empty sequence of sets containing vertices, can be some structure $\mathbb{T}$ parameterized on sets. And instead of choosing a 'path' in such a network, we can think of

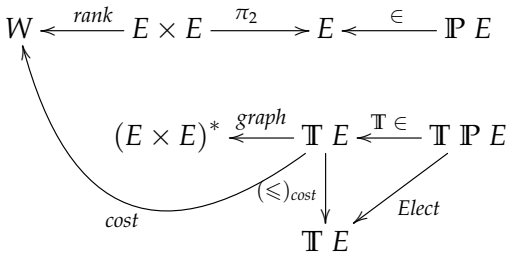
the problem as selecting, or electing, proxies for the sets in the structure. This generalized version of the problem is specified as follows:

$$Elect = \mathbb{T} \in \restriction (\leqslant)_{cost}$$

where

$$\mathbb{T}\ X \ \cong \ \mathbb{B}\ (X, \mathbb{T}\ X)$$

is a type functor for a recursively defined structure containing elements of type X, possibly under some conditions. This type functor is defined as being isomorphic to a specific bifunctor, allowing for the application of bifunctor laws in the derivation. The $\mathbb{T}\ \in$ tells us that the topology of the structure remains the same provided all sets of Es are non-empty, which is also assumed in the original problem. For instance, this structure can be a tree representing some hierarchical organization, where each node is a group of individuals in a team, and there is a need to pick a team leader. The optimizing criterion $(\leqslant)_{cost}$, as in the original problem, must utilize a notion of adjacency, with pairs of elements being attributed a score. In the following type diagram, with basic types E(ntity) and Weight,



the *graph* function generates an adjacency graph, and *rank* is the cost function per element pair. In the case of a hierarchical organization, there may be many ways to judge the cost of electing team leaders. A hierarchy can be represented by a nonempty rose tree:

$$RTree\ A = A + A \times (RTree\ A)^{+}$$
$$\mathsf{in} = [tip, node]$$

Suppose that for each pair of employees an interpersonal friction score is attributed, and *cost* considers only links between immediate superiors and their subordinates. Using Haskell notation:

$$\begin{cases} cost\ (Tip\ a) = 0 \\ cost\ (Node\ a\ xs) = sum\ [rank\ a\ (atRoot\ x) \mid x \leftarrow xs] \end{cases}$$

where *atRoot* is the function giving the element at the root of the tree, regardless of it being a tip or a node. This type of relation arises in the study of generalized notions of membership (Barbosa and Oliveira, 2006). For this specific structure, *atRoot* satisfies the following equation:

$$atRoot \cdot \mathsf{in} = [id, \pi_1] \tag{6.2}$$

DERIVATION The crux of the matter is the choice of relation Q involved in the application of thinning laws, with the guarantee that when thinning is eliminated (5.18), correctness is preserved.

$$
\begin{array}{ll}
& \mathbb{T} \in \upharpoonright (\leqslant)_{cost} \\
= & \quad \{ \text{ definition of } \mathbb{T} \text{ (3.4) } \} \\
& (\!| \mathsf{in} \cdot \mathbb{B}\ (\in, id) |\!) \upharpoonright (\leqslant)_{cost} \\
= & \quad \{ \text{ introduce thinning (5.27) } \} \\
& \in \upharpoonright (\leqslant)_{cost} \cdot (\!| \mathsf{in} \cdot \mathbb{B}\ (\in, \in) \downharpoonright Q |\!) \\
\supseteq & \quad \{ \ (\leqslant)_{cost} \text{ is connected so refine shrinking to the } minlist \text{ function } \} \\
& minlist\ (\leqslant)_{cost} \cdot (\!| \mathsf{in} \cdot \mathbb{B}\ (\in, \in) \downharpoonright Q |\!)
\end{array}
$$

Isolating the thinning part, we reason:

$$
\begin{array}{ll}
& \mathsf{in} \cdot \mathbb{B}\ (\in, \in) \downharpoonright Q \\
\supseteq & \quad \{ \text{ bifunctors; thinning partitioning rule (5.17) } \} \\
& \mu \cdot \mathbb{P}\ (\mathsf{in} \cdot \mathbb{B}\ (id, \in) \downharpoonright Q) \cdot \Lambda \mathbb{B}\ (\in, id) \\
\supseteq & \quad \{ \text{ thinning elimination (5.18) } \} \\
& \mu \cdot \mathbb{P}\ (\eta \cdot (\mathsf{in} \cdot \mathbb{B}\ (id, \in) \upharpoonright (\leqslant)_{cost})) \cdot \Lambda \mathbb{B}\ (\in, id) \\
= & \quad \{ \text{ (2.157) } \} \\
& \mathbb{P}\ (\mathsf{in} \cdot \mathbb{B}\ (id, \in) \upharpoonright (\leqslant)_{cost}) \cdot \Lambda \mathbb{B}\ (\in, id) \\
= & \quad \{ \text{ (2.193) } \} \\
& \mathbb{P}\ ((\in \upharpoonright (\leqslant)_{cost}) \cdot \Lambda(\mathsf{in} \cdot \mathbb{B}\ (id, \in))) \cdot \Lambda \mathbb{B}\ (\in, id) \\
\supseteq & \quad \{ \ (\leqslant)_{cost} \text{ is connected so refine shrinking to the } minlist \text{ function } \} \\
& \mathbb{P}\ (minlist\ (\leqslant)_{cost} \cdot \Lambda(\mathsf{in} \cdot \mathbb{B}\ (id, \in))) \cdot \Lambda \mathbb{B}\ (\in, id)
\end{array}
$$

The choice of $(\leqslant)_{cost}$ as the shrinking criterion in the thinning elimination step signals the intention to prove that we can partition the problem into multiple instances of the same problem, allowing us to use, for each partition, the same optimization criterion as the one used to obtain the optimal complete solution.

The Q relation will determine how this partition is made. We wish to partition the problem in such a way that only a single solution per element in the set at the root of the tree is "passed on" to larger problems. This can be done by setting

$$Q = (\leqslant)_{cost} \cap \frac{atRoot}{atRoot}$$

determining that when two trees share their root element ($b\ (\frac{atRoot}{atRoot})\ a \ \Leftrightarrow\ atRoot\ b = atRoot\ a$), only the one with the lower cost is needed to generate the optimal solution for the entire tree.

To prove that this logic works, it suffices to prove the conditions arising from introducing (5.27) and eliminating (5.18) thinning, which give upper and lower bounds for Q:

$$(\leqslant)_{cost} \cap \mathrm{img}\ (\mathsf{in} \cdot \mathbb{B}\ (id, \in)) \subseteq Q \subseteq (\leqslant)_{cost}$$

Given the definition of Q, it is trivially true that $Q \subseteq (\leqslant)_{cost}$. The proof of the rest of the statment makes use of $atRoot$ property (6.2):

$$\begin{array}{ll}
& (\leqslant)_{cost} \cap \mathrm{img}\ (\mathsf{in} \cdot \mathbb{B}\ (id, \in)) \subseteq (\leqslant)_{cost} \cap \dfrac{atRoot}{atRoot} \\
\Leftarrow & \quad \{\ \text{monotonicity (2.29)}\ \} \\
& \mathrm{img}\ (\mathsf{in} \cdot \mathbb{B}\ (id, \in)) \subseteq \dfrac{atRoot}{atRoot} \\
\equiv & \quad \{\ \text{symmetric division (2.112); shunting (2.58, 2.59); definition of } \mathrm{img} \cdot \text{ (2.39); converse}\ \} \\
& atRoot \cdot \mathsf{in} \cdot \mathbb{B}\ (id, \in) \cdot \mathbb{B}\ (id, \in^{\circ}) \cdot (atRoot \cdot \mathsf{in})^{\circ} \subseteq id \\
\equiv & \quad \{\ \text{(6.2) (twice)}\ \} \\
& [id, \pi_1] \cdot \mathbb{B}\ (id, \in) \cdot \mathbb{B}\ (id, \in^{\circ}) \cdot [id, \pi_1]^{\circ} \subseteq id \\
\equiv & \quad \{\ \text{bifunctors; direct sum absorption (2.87); biproduct absorption (2.85)}\ \} \\
& id \cup \pi_1 \cdot (id \times \in \cdot \in^{\circ}) \cdot \pi_1^{\circ} \subseteq id \\
\Leftarrow & \quad \{\ \text{relational join (2.14); free theorem of } \pi_1 \text{ (2.72)}\ \} \\
& \pi_1 \cdot \pi_1^{\circ} \subseteq id \\
\equiv & \quad \{\ \pi_1 \text{ is simple (2.41)}\ \} \\
& true \\
\Box &
\end{array}$$

The final program

$$gln = minlist\ (\leqslant)_{cost} \cdot (\!|\, \mathbb{P}\ (minlist\ (\leqslant)_{cost} \cdot \Lambda(\mathsf{in} \cdot \mathbb{B}\ (id, \in))) \cdot \Lambda\mathbb{B}\ (\in, id)\, |\!) \tag{6.3}$$

is better understood by separating the catamorphism's gene into a coproduct of functions $base$ and $step$:

$$\begin{array}{ll}
& \mathbb{P}\ (minlist\ (\leqslant)_{cost} \cdot \Lambda(\mathsf{in} \cdot \mathbb{B}\ (id, \in))) \cdot \Lambda\mathbb{B}\ (\in, id) \\
= & \quad \{\ \text{definitions of } \mathsf{in} \text{ and } \mathbb{B}\ \} \\
& \mathbb{P}\ (minlist\ (\leqslant)_{cost} \cdot \Lambda([tip, node] \cdot (id + id \times \in))) \cdot \Lambda(\in + \in \times id) \\
= & \quad \{\ \text{direct sum absorption (2.87)}\ \} \\
& \mathbb{P}\ (minlist\ (\leqslant)_{cost} \cdot [\Lambda tip, \Lambda(node \cdot (id \times \in))]) \cdot \Lambda(\in + \in \times id) \\
= & \quad \{\ \text{function transpose (2.234); coproduct fusion (2.140); } minlist\ R \cdot \eta = id\ \} \\
& \mathbb{P}\ [tip, minlist\ (\leqslant)_{cost} \cdot \Lambda(node \cdot (id \times \in))] \cdot \Lambda(\in + \in \times id)
\end{array}$$

$$
\begin{array}{ll}
= & \{\ (2.153);\ \Lambda\in = id\ (2.140)\ \} \\
& [\mathbb{P}\ tip, \mathbb{P}\ (minlist\ (\leqslant)_{cost} \cdot \Lambda(node \cdot (id \times \in))) \cdot \Lambda(\in \times id)]
\end{array}
$$

The program is now implemented in Haskell representing sets as (nonempty) lists. This means the power relator $\mathbb{P}$ is implemented by fmap on lists, and $\Lambda(node \cdot (id \times \in))$ and $\Lambda(\in \times id)$ by appropriate list comprehensions.

$$
\begin{array}{l}
\Lambda(\in \times id)\ (s, tss) = [(a, tss) \mid a \leftarrow s] \\
\Lambda(node \cdot (id \times \in)))\ (a, tss) = [node\ (a, ts) \mid ts \leftarrow tss]
\end{array}
$$

These equalities follow from the equivalent of laws (2.240) and (2.241) for lists.

$$
\begin{array}{l}
r\ a\ b = cost\ a \leqslant cost\ b \\
cost\ (Tip\ a) = 0 \\
cost\ (Node\ a\ xs) = sum\ [rank\ a\ (atRoot\ x) \mid x \leftarrow xs] \\
atRoot = [id, \pi_1] \cdot \mathsf{out} \\
gln = minlist\ r \cdot (\!|\ [base, step]\ |\!)\ \mathbf{where} \\
\quad base = \mathsf{fmap}\ tip \\
\quad step\ (s, tss) = \mathsf{fmap}\ minExtend\ [(a, tss) \mid a \leftarrow s] \\
\quad\quad minExtend\ (a, tss) = minlist\ r\ [node\ (a, ts) \mid ts \leftarrow tss]
\end{array}
$$

6.3 SUMMARY

In this chapter, a characterization of Dynamic Programming encompassing the concept of relational thinning was proposed. A few types of algorithms were identified, ones in which thinning is not possible, others where the problem is partitioned into multiple maximization subproblems, and then two methods of thinning: 'decomposition' thinning and 'explicit' thinning. Lastly, two case studies were developed, aiming to generalize their treatment in the current literature.

7

CONCLUSIONS

The dawn of the modern computer era led to a boom in algorithm research. Computer scientists strove not only to devise algorithms that work, but also to understand their structure. Formal methods in software engineering achieve this by using mathematics both as a human and a machine language, putting theoretical understanding in service of practical application.

Using the Algebra of Programming framework based on a pointfree relation calculus, the initial purview of this work was investigating the importance of the metaphorism specification pattern (Oliveira, 2018) in Dynamic Programming. However, the focus of this work shifted. The existence of problems straddling the border between DP and the greedy approach, such as the coin changing problem, called for a broader exploration of Dynamic Programming and the greedy algorithm. In both problem spaces, the importance of shrinking as a problem specification device, as well as an algorithm implementation one, ought be underscored. The theory underlying the shrinking relational combinator is the foundation this work is built upon.

Being a field in which the greedy algorithm has been extensively studied, matroid theory provided an ample basis for this project. In specific, the algorithmic properties of matroids in relation to the greedy algorithm lay out a path to synthesize algorithms for problems exhibiting matroidal structures. To this effect, concepts from matroid theory were expressed in relation algebra as statements and their sufficient conditions, and these were used in the derivation of a greedy algorithm.

Dynamic Programming was framed as a middle ground between brute force algorithms and the greedy algorithm. This approach follows naturally from the specification of and derivation techniques for optimization problems, and motivates and exploration of thinning, a technique used to cull the solution space, keeping only useful solutions.

7.1 SUMMARY OF CONTRIBUTIONS

This work's main contributions span from the very broad to the very specific. It aims to contextualize the study of Dynamic Programming in relation to the greedy algorithm, investigating the concept of thinning through the thinning relational operator, suggested by Oliveira (2018) for the potential interaction with metaphorisms. This was the original focus of this work, specifically the usefulness of the metaphorism specification pattern in Dynamic Programming. Although useful in the specification of some problems, this pattern was not found to be immediately generalizable.

Initially, a few inductive relations on lists were introduced, along with auxiliary laws which proved useful throughout this work. The sublist ($\trianglelefteq$) relation was defined in terms of the subsequence ($\sqsubseteq$) and permutation $Perm$ relations. The $Select$ and $SelectBy$ relations are especially important for their role in the greedy algorithm on lists.

Then, using matroid theory, a few properties of set systems were expressed in relational algebra, as well as statements pertaining to the greedy algorithm. Sufficient conditions were given for some of these statement to hold, and subsequently proven as part of the derivation of the greedy algorithm for a simple problem — the "color problem".

In preparation for tackling Dynamic Programming, the theory underlying the thinning relational operator was extended. Basic properties of thinning were proven, as well as introduction and elimination laws interacting with shrinking, laws for preorder thinning and for thinning over recursive structures.

Finally, a characterization of DP algorithms was made. Not all such algorithms involve thinning. They may need to search the entire solution space, in which case thinning is not possible, or they can be partitioned into multiple maximization problems, making thinning unnecessary. When thinning is useful, it comes in two kinds. 'Decomposition' thinning focuses on which problem decompositions can be ignored, while 'explicit' thinning relies on a modified criterion to exclude generated partial solutions after each step.

Two Dynamic Programming case studies, the knapsack problem and a generalization of the "paths in a layered network" problem, were developed. The first is a case of explicit thinning, while the second is a partitioning algorithm. Both their derivations made use of the thinning operator and its laws.

7.2 FUTURE WORK

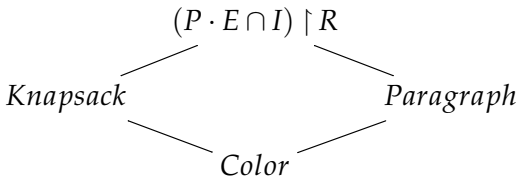
The statement in (4.16) was proven by informal means. Finding an elegant proof is left as future work, helping to further develop a calculational approach to matroid theory and the greedy algorithm.

A deeper exploration of matroid theory in relation-algebraic terms is welcome. This work focuses on introducing elementary notions of matroid theory, and restricts itself to the case of the greedy algorithm on lists under a generic matroid with a stable linear objective function. It would be interesting to investigate other set system structures (e.g. greedoids) under more complex objective functions, and to generalize the formulation of matroids to arbitrary container types (Hoogendijk and de Moor, 2000).

The color problem, specified in four parts — recall equation (4.17)

$$M = (P \cdot E \cap I) \upharpoonright R$$

— instantiates a scheme that also encapsulates other problems. In the knapsack problem, there is no invariant, and so $I = \top$, while the paragraph problem can be specified as a post-conditioned metaphorism — $E = \top$ and $I = \frac{(\!|f|\!)}{(\!|g|\!)}$. This suggests the following problem hierarchy



possibly leading to a wider and more comprehensive *taxonomy* of algorithmic problems, provided general specification schemes such as (4.17) and corresponding hierarchies are further explored and put together.

On a more technical level, a few aspects were disregarded to focus on other areas. For instance, while the subset relation

$$Subset = \in \setminus \in$$

can be expressed as a relational division of set memberships, the sublist relation $(\trianglelefteq)$ is not a division of list memberships $\epsilon \setminus \epsilon$. Instead, $(\trianglelefteq)$ is expressed as a hylomorphism of two different relations. Is there a more useful definition of this relation, one that could potentially be generalized to other container structures?

The problem specifications in the problem catalogue often include a restriction of the solution space by way of a post-condition ϕ_p. Perhaps a more convenient way to express this restriction would be as shrinking the solution space (2.197)

$$S \upharpoonright (\phi_q \cdot \top) = \phi_q \cdot S \quad \Leftarrow \quad S \text{ is entire}$$

suggesting that shrinking laws and theorems may be useful in wider contexts. For example, the "greedy post-conditioning rule" (4.7) could be derived by promoting shrinking into the catamorphism instead of cata fusion.

Another subject left unexplored is the relationship between the tail-recursive implementation of the greedy algorithm on lists, and the one used in this work, a hylomorphism on lists. The structure used here conceptualizes the greedy algorithm as selecting the worst element to add last, while the typical formulation of the algorithm is a while loop (tail-recursive hylomorphism) which builds a list by making the locally optimal choice — selecting the best element to add first. The two algorithms are operationally equivalent and future research could focus on proving this equivalence. The theory of adjoint folds (Hinze, 2013) seems particularly well equipped to handle this.

As this work has a focus on the distinction between Greedy and Dynamic Programming, and on the theory of thinning, a formalization of tabulation methods, which are also tackled by Bird and de Moor (1997), although originally part of the work plan, is left as future work.

Outside of the scope of this work falls the work of developing tools to automate the synthesis of algorithmic solutions for greedy and DP problems. This can be done by leveraging the AoP framework and its extensions, as shown by Silva and Oliveira (2008), including through the use of proof assistants (Mizoguchi et al., 2016). In

particular, the AoPA library developed by Chiang and Mu (2016) encodes AoP styled program derivation (both functional and relational) in Agda and seems particularly attractive for such purposes.

BIBLIOGRAPHY

L.S. Barbosa and J.N. Oliveira. Transposing partial components—an exercise on coalgebraic refinement. *Theor. Comput. Sci.*, 365:2–22, 11 2006. doi: 10.1016/j.tcs.2006.07.030.

L.S. Barbosa, J.N. Oliveira, and A.M. Silva. Calculating Invariants as Coreflexive Bisimulations. In *AMAST'08*, volume 5140 of *LNCS*, pages 83–99. Springer-Verlag, 2008.

R. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8): 716–719, 1952. ISSN 0027-8424. doi: 10.1073/pnas.38.8.716. URL https://www.pnas.org/content/38/8/716.

R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, 11 1954. URL https://projecteuclid.org:443/euclid.bams/1183519147.

R. Bellman. Some directions of research in dynamic programming. *Unternehmensforschung*, 7(3):97–102, 1963. doi: 10.1007/BF01923711. URL https://doi.org/10.1007/BF01923711.

R. Bellman. *Dynamic Programming*. Princeton University Press, USA, 2010. ISBN 0691146683.

R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall, 1997.

R. Bird and J. Gibbons. *Algorithm Design with Haskell*. Cambridge University Press, 2020. doi: 10.1017/9781108869041.

R. Bird and S.-C. Mu. A greedy algorithm for dropping digits. *Journal of Functional Programming*, 31:e29, 2021. doi: 10.1017/S0956796821000198.

R. Bird and F. Rabe. How to calculate with nondeterministic functions. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 138–154, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33636-3.

R. Bird, J. Gibbons, and S.-C. Mu. Algebraic methods for optimization problems. pages 281–308, 01 2000. ISBN 978-3-540-43613-3. doi: 10.1007/3-540-47797-7_8.

E.A. Boiten. Improving recursive functions by inverting the order of evaluation. *Science of Computer Programming*, 18(2):139 – 179, 1992. ISSN 0167-6423. doi: https://doi.org/10.1016/0167-6423(92)90008-Y. URL http://www.sciencedirect.com/science/article/pii/016764239290008Y.

X. Cai. Canonical coin systems for change-making problems, 2009.

Y.-H. Chiang and S.-C. Mu. Formal derivation of greedy algorithms from relational specifications: A tutorial. *J. Log. Algebraic Methods Program.*, 85(5):879–905, 2016. doi: 10.1016/j.jlamp.2015.12.003. URL https://doi.org/10.1016/j.jlamp.2015.12.003.

D. Cofer. Formal methods in the aerospace industry: Follow the money. In Toshiaki Aoki and Kenji Taguchi, editors, *Formal Methods and Software Engineering*, pages 2–3, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-34281-3.

B. Cohen. A brief history of formal methods. *Formal Aspects of Computing*, 01 1995.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844.

S.A. Curtis. The classification of greedy algorithms. *Science of Computer Programming*, 49(1):125–157, 2003. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2003.09.001. URL https://www.sciencedirect.com/science/article/pii/S0167642303000340.

P.J. Freyd and A. Scedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North-Holland, 1990.

I. Guimarães. Apresento-vos os... matróides!!! https://www.youtube.com/watch?v=7NZO9fgSED8, 2021. MathGurl Youtube channel. Accessed: 2021-10-08.

I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4):1–36, 2007.

P. Helman, B. Moret, and H. Shapiro. An exact characterization of greedy structures. *SIAM Journal on Discrete Mathematics*, 6, 03 1995. doi: 10.1137/0406021.

R. Hinze. Adjoint folds and unfolds—an extended study. *Science of Computer Programming*, 78:2108–2159, 11 2013. doi: 10.1016/j.scico.2012.07.011.

P.F. Hoogendijk and O. de Moor. Container types categorically. *Journal of Functional Programming*, 10(2): 191–225, 2000. URL citeseer.nj.nec.com/hoogendijk00container.html.

P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Comput. Surv.*, 21 (3):359–411, sep 1989. ISSN 0360-0300. doi: 10.1145/72551.72554. URL https://doi.org/10.1145/72551.72554.

C. Kern and M.R. Greenstreet. Formal verification in hardware design: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 4(2):123–193, apr 1999. ISSN 1084-4309. doi: 10.1145/307988.307989. URL https://doi.org/10.1145/307988.307989.

B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988. ISBN 0131103709.

T. Kohn and G. van Rossum. Structural pattern matching: Motivation and rationale. PEP 635, 2020. URL https://www.python.org/dev/peps/pep-0635/.

B. Korte and L. Lovász. Greedoids - a structural framework for the greedy algorithm. 1984.

Y.A. Liu and S.D. Stoller. Dynamic programming via static incrementalization. In *Proceedings of the 8th European Symposium on Programming Languages and Systems*, ESOP '99, page 288–305, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3540656995.

P. Mateti. Peter naur's "telegram problem". https://web.archive.org/web/20140906085522/http://cecs.wright.edu/people/faculty/pmateti/Courses/7140/Lectures/TelegramProblem/telegram-problem.html, 2013.

C.A. Meadows. Formal verification of cryptographic protocols: A survey. In Josef Pieprzyk and Reihanah Safavi-Naini, editors, *Advances in Cryptology — ASIACRYPT'94*, pages 133–150, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49236-8.

Y. Mizoguchi, H. Tanaka, and S. Inokuchi. Formalization of proofs using relational calculus. In *2016 International Symposium on Information Theory and Its Applications (ISITA)*, pages 527–531, 2016.

A. Morihata, M. Koishi, and A. Ohori. Dynamic programming via thinning and incrementalization. In Michael Codish and Eijiro Sumii, editors, *Functional and Logic Programming*, pages 186–202, Cham, 2014. Springer International Publishing. ISBN 978-3-319-07151-0.

S.-C. Mu. On building trees with minimum height, relationally. pages 153–162, 01 2000.

S.-C. Mu and J.N. Oliveira. Programming from Galois connections. *JLAP*, 81(6):680–704, 2012.

S.-C. Mu, Y.-H. Lyu, and A. Morihata. Constructing datatype-generic fully polynomial-time approximation schemes using generalised thinning. In *Proceedings of the 6th ACM SIGPLAN Workshop on Generic Programming*, WGP '10, page 97–108, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302517. doi: 10.1145/1863495.1863508. URL https://doi.org/10.1145/1863495.1863508.

C. Okasaki. Functional data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. doi: 10.1201/9781420035179.pt6. URL https://doi.org/10.1201/9781420035179.pt6.

J.N. Oliveira. Monads arising from membership. Technical Report TR-HASLab:02:2016, INESC TEC & U.Minho, Gualtar Campus, Braga, May 2016.

J.N. Oliveira. Programming from metaphorisms. *JLAMP*, 94:15–44, January 2018. ISSN 2352-2208. doi: https://doi.org/10.1016/j.jlamp.2017.09.003. URL http://www.sciencedirect.com/science/article/pii/S2352220817301645. (DOI).

J.N. Oliveira. Program Design by Calculation, 2019. Draft of textbook in preparation, current version: October 2019. Informatics Department, University of Minho (PDF).

J.N. Oliveira and M.A. Ferreira. Alloy meets the algebra of programming: A case study. *IEEE Trans. Soft. Eng.*, 39(3):305–326, 2013.

A. Pettorossi. A powerful strategy for deriving efficient programs by transformation. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84, page 273–281, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0897911423. doi: 10.1145/800055.802044. URL https://doi.org/10.1145/800055.802044.

P. Silva and J.N. Oliveira. 'galculator': functional prototype of a galois-connection based proof assistant. pages 44–55, 07 2008. doi: 10.1145/1389449.1389456.

A. Vince. A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1):247–260, 2002. ISSN 0166-218X. doi: https://doi.org/10.1016/S0166-218X(01)00362-6. URL https://www.sciencedirect.com/science/article/pii/S0166218X01003626.

D.J.A. Welsh and London Mathematical Society. *Matroid Theory*. L.M.S. monographs. Academic Press, 1976. ISBN 9780127440507. URL https://books.google.pt/books?id=cfXuAAAAMAAJ.

Part III

APPENDICES

A

PROBLEM CATALOGUE

Here follows a curated collection of DP problems, including combinatorial optimization problems as well as problems specified by a recurrence relation. Each problem description is accompanied by a formal specification, and the corresponding type diagram where possible, followed by a brief discussion of the methods used to derive functional implementations of the specification.

A.1 THE COLOR PROBLEM

DESCRIPTION Consider a set of items S, each with a value and a measure of some kind, for example, their color. Select the set of items maximizing the total value, while making sure that no two items have the same color, and that all the colors in the given set S are preserved in the resulting set.

The problem is optimally solved by the greedy algorithm: choose the item with the highest value, and include it if there is no item with the same color in the set; repeat this process until there are no more items to consider.

RELATIONAL SPECIFICATION It is worth considering this simple problem because its description features some important concepts in algorithm design. First, we specify the solution space which is, in this case, the subsets of S, with sets being encoded by finite lists:

$$E = (\trianglelefteq)$$

Then we impose the post-condition that no colors can repeat in the output sets

$$Q = \phi_{nr\,c} \textbf{ where } nr\,f = noRepeats \cdot \mathsf{map}\,f$$

and the invariant stating the preservation of the color set:

$$I = \frac{\Lambda Cs}{\Lambda Cs} \textbf{ where } Cs = \epsilon \cdot \mathsf{map}\,c$$

Finally we choose the optimization relation:

$$R = (\geqslant)_{value} \textbf{ where } value = sum \cdot \mathsf{map}\,v$$

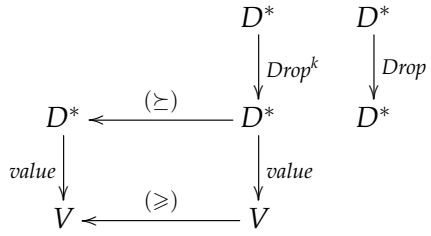
These four parts combined give us a relational specification that can be suited to describe many problems:

$$P = (Q \cdot E \cap I) \upharpoonright R$$

A.2 DROPPING DIGITS

DESCRIPTION Taken from (Bird and Mu, 2021)[1], the "Dropping Digits" problem consists in finding a way to drop k digits from a natural number of at least k digits, maximizing the final value. This optimization problem, despite not having direct practical applications to date, is interesting to study in the realm of problems solved by the greedy algorithm.

TYPE DIAGRAM Representing a number by a list of digits (D^*) from which a value of type V can be calculated:



SPECIFICATION The solution space is obtained by repeatedly applying a $Drop$ operation k times, removing k elements from the list which represents the number and preserving the order of the rest. The optimization criterion is the lexicographic ordering on the list $(\succeq)$, which is equivalent to comparing the values of different solutions through $(\geqslant)_{value}$.

$$MaxDrop\ k = Drop^k \upharpoonright (\succeq)$$

where $Drop = Discard \cap (\sqsubseteq)$. Perhaps in a surprising way, this problem can be solved through a relatively simple greedy algorithm, as shown by Bird and Mu (2021): the best digit to drop next, guaranteeing that the highest number is obtained, is this first digit, starting from the most significant, which is lower than next one. So, for example, the next digit to drop in 69732 would be 6, obtaining 9732; if no digit satisfies this condition — the list is in descending order — then drop the least significant one. After 9732, we would drop the digit 2 and get 973.

This is an example of a problem which does not follow the "Better-Global" principle. Simply arriving at a better partial solution is not enough to prove that the completed solution will be better. Instead, only the "Best-Global" principle applies: only by making the optimal choice at each step can we make sure that the optimal solution is obtained.

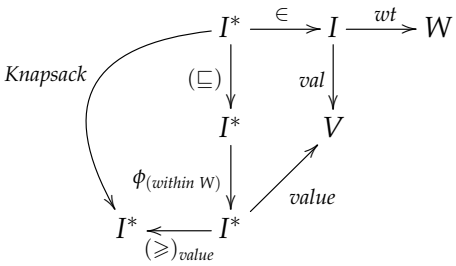
1 Where it is inspired by https://leetcode.com/problems/remove-k-digits/.

A.3 THE KNAPSACK PROBLEM

DESCRIPTION The knapsack problem is ubiquitous in discussions of dynamic programming. Here we formulate the 0-1 knapsack problem.

Consider a set of N items, two functions, respectively named wt and val, that give the weight and the value of a given item, and a weight limit W. The value/weight of a set of items is the sum of the value/weight of each of its items. Determine the subset of items with the highest value, while still abiding by the weight limit, that is, its weight should be no greater than W.

TYPE DIAGRAM Let I(tem) and V(alue) be basic types in:



SPECIFICATION It is clear that the optimizing relation should rate higher-valued sets above lower-valued sets. So, the problem can be specified by

$$
\begin{aligned}
&Knapsack = (\phi_{within\ W} \cdot (\sqsubseteq)) \upharpoonright (\geqslant)_{value} \ \mathbf{where} \\
&\quad (\sqsubseteq) = (\!|\, [nil, cons \cup \pi_2] \,|\!) \\
&\quad value = sum \cdot \mathsf{map}\ val \\
&\quad within\ W\ x = weight\ x \leqslant W \\
&\quad weight = sum \cdot \mathsf{map}\ wt \cdot
\end{aligned}
$$

The $(\sqsubseteq)$ relation generates all possible subsequences, recall (2.211). It is then post-conditioned by predicate $within\ W\ x = (weight\ x \leqslant W)$, where $weight = sum \cdot \mathsf{map}\ wt$, which restricts the search to only those subsequences that do not exceed the weight limit.

A.4 FIBONACCI NUMBERS

DESCRIPTION The Fibonacci sequence was introduced in Europe by Italian mathematician Leonardo Bonacci, or simply Fibonacci. The sequence starts

$$1, 1, 2, 3, 5, 8, 13, 21, 34, ...$$

and while there is a closed formula to compute a Fibonacci number at a particular position, known as Binet's formula, different algorithms to do so are instructive for beginners and interesting as case studies in algorithm research.

SPECIFICATION The problem of computing the i^{th} number in the Fibonacci sequence, where $i \geqslant 0$, can be specified through a recursive mathematical function:

$$fib(n) = \begin{cases} 1 & \Leftarrow \ n = 0 \\ 1 & \Leftarrow \ n = 1 \\ fib(n-1) + fib(n-2) & \Leftarrow \ n > 1 \end{cases}$$

Obviously, this can be directly translated into the following program:

$$\begin{aligned} &fib\ 0 = 1 \\ &fib\ 1 = 1 \\ &fib\ n = fib\ (n-1) + fib\ (n-2) \end{aligned}$$

However, this is a naive implementation that has exponential time complexity ($fib(n) = \Theta(\phi^n)$). One way to optimize it is to use memoization, which reduces time complexity to $\Theta(n)$, but requires $\Theta(n)$ space. Using tabulation renders the same time and space complexity since a structure of size n is needed. But since computing the i^{th} Fibonacci number depends (directly) only on a constant number of previous results — two —, we can improve on the tabulation approach by only keeping the two previous results at all times, yielding the following program running in linear time and constant space:

$$\begin{aligned} &fib\ n = m\ \mathbf{where} \\ &\quad (\_, m) = fib'\ (n) \\ &\quad fib'\ 0 = (1, 1) \\ &\quad fib'\ n = (x + y, x)\ \mathbf{where}\ (x, y) = fib'\ (n-1) \end{aligned}$$

This version is easily calculated by a program calculation technique known as *mutual recursion* or *tupling* (Bird and de Moor, 1997; Oliveira, 2019).

A.5 BINOMIAL COEFFICIENT

DESCRIPTION The binomial theorem gives a formula for expanding a polynomial $(x + y)^n$ as a sum of terms ax^by^c, where $b + c = n$ and a is the binomial coefficient $\binom{n}{k}$, with $k = b$ or $k = c$, which result in the same value. A binomial coefficient $\binom{n}{k}$ is also the kth entry in the nth row (both starting from 0) of Pascal's triangle.

SPECIFICATION Given integers n and $k \leqslant n$, there are multiple ways the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

can be recursively defined. The simplest is obtained by fixing n and doing primitive recursion on k, by factorial decomposition:

$$\binom{n}{0} = 1$$
$$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$$

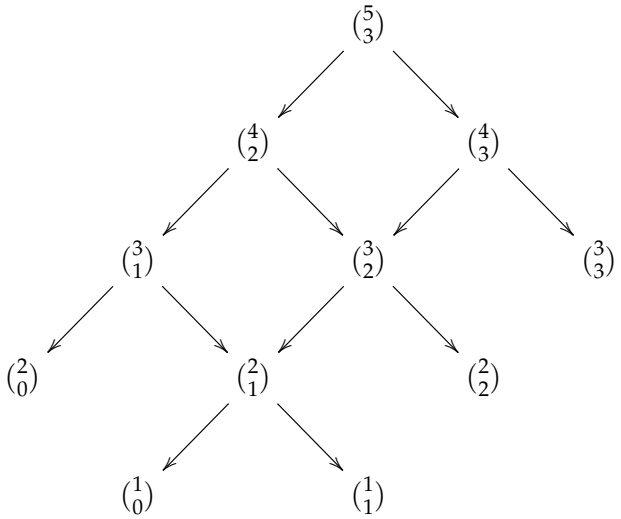
The multiplicative term $\frac{n-k}{k+1}$ is obtained by dividing $\binom{n}{k+1}$ by $\binom{n}{k}$ using the closed formula, leading to an efficient solution without overlapping subproblems (a list hylomorphism that generates all $\frac{n-k}{k+1}$ factors and then multiplies them all). Another recursive formula, interesting for the existence of overlapping subproblems and where DP can be applied, is as follows:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \Leftarrow \quad 1 \leqslant k \leqslant n-1$$
$$\binom{n}{0} = \binom{n}{n} = 1 \quad \Leftarrow \quad n \geqslant 0$$

This gives the inefficient recursive program:

$$\begin{aligned}
&\mathit{binom}\ _\ 0 = 1\\
&\mathit{binom}\ n\ k\\
&\quad \mid n \equiv k = 1\\
&\quad \mid \mathit{otherwise} = \mathit{binom}\ (n-1)\ (k-1) + \mathit{binom}\ (n-1)\ k
\end{aligned}$$

The recurrence relation in the specification gives rise to the following call graph, for $\binom{5}{3}$:



Boiten (1992) derives an efficient tabulation algorithm by traversing the call graph in a linear order.

A.6 PLANNING A COMPANY PARTY

DESCRIPTION The following problem is sourced from (Cormen et al., 2009), where it appears as an exercise in their chapter on dynamic programming:

> Professor Stewart is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.
>
> (...) Describe an algorithm to make up a guest list that maximizes the sum of the conviviality ratings of the guests.

TYPE DIAGRAM Let E(mployee), R(ating), $Tree\ A = A \times (Tree\ A)^*$ be types. With abbreviation $S = \langle include, Exclude \rangle$, we have:

$$\begin{array}{ccccc} E & \xleftarrow{(\leqslant)_{rating}} & E & \xleftarrow{\in} & E^* \\ & {\scriptstyle rating}\downarrow & & {\scriptstyle (\leqslant)_{ratings}}\downarrow & \\ & \mathbb{R} & & E^* & \end{array} \qquad \begin{array}{ccccc} Tree\ E & \xleftarrow{node} & E \times (Tree\ E)^* \\ {\scriptstyle (\!|\, S \,|\!)}\downarrow & & \downarrow{\scriptstyle \mathbb{F}\,(\!|\, S \,|\!)} \\ E^* & \xleftarrow{Choose} & E^* \times E^* & \xleftarrow[S]{} & E \times (E^* \times E^*)^* \end{array}$$

SPECIFICATION Bird and de Moor (1997) derive a dynamic programming algorithm starting from the following specification:

$$Party \upharpoonright (\geqslant)_{ratings}$$

where $E^* \xleftarrow{Party} Tree\ E$ is the relation that generates all possible sets of party attendees, and $(\geqslant)_{ratings}$ — for

$$ratings = sum \cdot \mathsf{map}\ rating$$

— determines that the sum of the conviviality rating of party attendees should be maximized.

A good approach to generating all possible sets of party attendees is too split the problem in two, first generating those sets including the immediate supervisor (the root of the hierarchy tree), then those excluding that person. This is done recursively down the hierarchical structure of the company, alternatively including or excluding an immediate supervisor to allow his or her employees to attend. Algebraically, this set can be described as follows:

$$Party = Choose \cdot (\!|\, \langle include, Exclude \rangle \,|\!)$$

The *Choose* relation, as the name suggests, chooses between including or excluding the root of the tree — the president of the company.

$$Choose = \pi_1 \cup \pi_2$$

The *include* and *Exclude* relations can also be expressed through familiar operators acting on lists:

$$include = cons \cdot (id \times (concat \cdot \mathsf{map}\ \pi_2))$$
$$Exclude = \pi_2 \cdot (id \times (concat \cdot \mathsf{map}\ Choose))$$

While *include* is a function, *Choose* and *Exclude* are not. However, it is possible to compute $\Lambda Choose$ and $\Lambda Exclude$, which are functions returning sets of sets of party attendees. Using this approach, Bird and de Moor (1997) derive a DP algorithm by the Greedy Theorem.

A.7 SHORTEST PATHS ON A CYLINDER

DESCRIPTION Consider an $n \times m$ array of positive integers, representing a cylinder on a horizontal axis. This means the top and bottom rows of the array are adjacent. A path in this cylinder is a sequence of m integers, each from the corresponding column, such that for each column after the first, the chosen integer must be one of the three that are adjacent to the integer in the previous column. The cost of a path is simply the sum of the chosen integers in the sequence. Find the path of least cost.

The problem can be specified by $Paths \upharpoonright (\leqslant)_{cost}$, where $Paths$ is the relation which generates all valid paths, and $(\leqslant)_{cost}$ specifies a cost order.

A.8 THE COIN CHANGE PROBLEM

DESCRIPTION Given a set of coins with various denominations, the coin change or change-making problem is to select the smallest number of coins to make change for a specific value. Unlike with individual items in the 0-1 knapsack problem, a similar combinatorial optimization problem, a coin can be selected an unlimited amount of times.

SPECIFICATION A combinatorial optimization problem can always be specified as the shrinking of a candidate-producing relation:

$$coinChange\ cs \subseteq Change\ cs \upharpoonright (\leqslant)_{\mathsf{length}}$$

where *Change* generates all possible ways to make change given a set of coins cs.

This is a well known case of a problem whose parameters determine what kind of solution exists: for 'canonical' coin systems (Cai, 2009), such as the US dollar or the Euro, it is always possible to use the greedy approach to

obtain the optimal solution — select the highest denomination coin below the amount of change left to make. For example, with the US dollar, in which the set of coin denominations is $\{1, 5, 10, 20\}$, it is easy to see that:

$$\mathit{coinChange}\ \{1, 5, 10, 20\}\ 32 = [20, 10, 1, 1]$$

However, in a coin system with denominations $\{1, 5, 15, 20, 50\}$,

$$\mathit{coinChange}\ \{1, 5, 15, 20\}\ 32 = [15, 15, 1, 1]$$

while the greedy algorithm would give the non-optimal solution $[20, 5, 5, 1, 1]$. When the greedy algorithm is not viable, a DP algorithm is needed. Here, one possible tabulation scheme is given. With input n, we build a list A of n solutions, starting from $A\ [0] = [\,]$ (no coins are needed). Then, for each denomination $d_j, 1 \leqslant j \leqslant k$ we build a new solution d_j places ahead of the current solution — $A[i + d_j] = d_j : A[i]$ where i is the current solution's index in the list. If the coin system covers every possible value, then, when $i = n$, $A\ [i]$ gives the list of coins to select, and $\mathsf{length}\ (A\ [i])$ is the minimum number of coins needed to make change for n. This tabulation scheme takes $O\ (n)$ space and $O\ (nk)$ time.

A.9 THE SECURITY VAN PROBLEM

DESCRIPTION Suppose a bank has a known sequence of deposits and withdrawals. For security reasons the total amount of cash in the bank should never exceed some fixed amount N, assumed to be at least as large as any single transaction. To cope with demand and supply, a security van can be called upon to deliver funds to the bank or to take away a surplus. The problem is to compute a schedule under which the van visits the bank a minimum number of times.

This problem can be solved by partitioning the sequences of transactions into *secure* subsequences. A sequence of transactions is secure if, assuming an existing supply of cash previously left by a security van, the bank is able to perform its transactions without running out of cash, exceeding the fixed amount N, or having to resort to a security van visit.

As such, the minimum amount of visits is given by the minimum length partition out of those made up of secure transaction sequences. Naturally, the problem can be specified as $(\mathsf{map}\ \mathit{secure}\ ?\ \cdot \mathit{Partition}) \upharpoonright (\leqslant)_{\mathsf{length}}$, with $(\leqslant)_{\mathsf{length}}$ comparing the length of the partitions.

A.10 PATHS IN A LAYERED NETWORK

DESCRIPTION A layered network is a non-empty sequence of sets of vertices. A path in a layered network is a sequence of vertices constructed by choosing a single vertex from each set, and its cost is the sum of the weight of each transition between adjacent vertices. Determine the path of least cost.

Figure 2 shows an example of a layered network, including the weights between vertices of adjacent layers. Note that each set may have a different number of vertices, and that each one is connected to every vertex of the

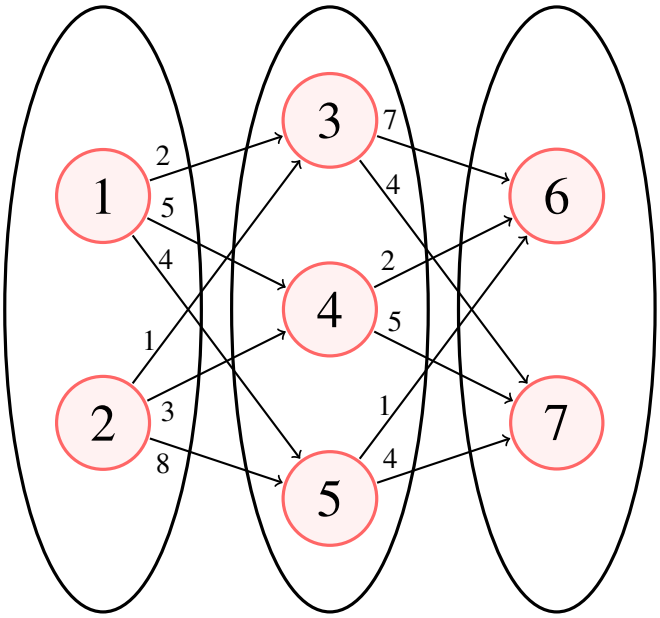
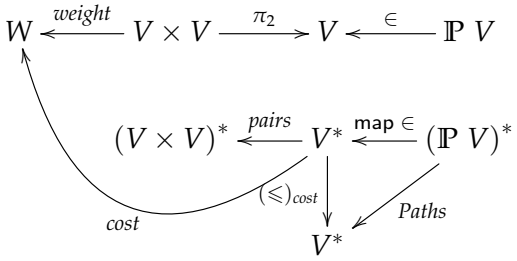


Figure 2.: Example illustration of a layered network

next layer, such that our choice is not constrained by previous ones, unlike, for instance, in the "shortest paths on a cylinder" problem.

TYPE DIAGRAM Let V(ertix) and W(eight) be basic types in:



RELATIONAL SPECIFICATION This description has a quite direct translation into a specification using the shrinking operator:

$$Paths = (\mathsf{map}\ \in) \upharpoonright (\leqslant)_{cost}$$

since $\mathsf{map}\ \in$ (for non-empty lists) relates the network to every possible path by literally choosing a vertex from each set in the list. And $(\leqslant)_{cost}$ compares paths by their cost, defined by the sum of the weights between each pair of adjacent vertices:

$$cost = sum \cdot \mathsf{map}\ weight \cdot pairs$$
$$pairs = zip \cdot \langle id, tail \rangle$$

where *tail* and *zip* are the standard functions (using Haskell syntax):

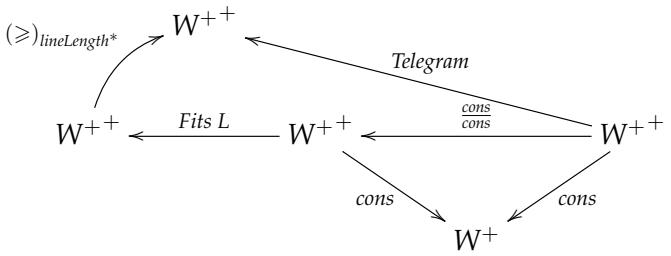
$$
\begin{aligned}
&tail\ (\_ : xs) = xs \\
&zip\ ([\,], \_) = [\,] \\
&zip\ (\_, [\,]) = [\,] \\
&zip\ ((x : xs), (y : yx)) = (x, y) : (zip\ (xs, yx))
\end{aligned}
$$

A.11 THE TELEGRAM PROBLEM

DESCRIPTION Originally described by Peter Naur, the telegram problem asks us to reorganize a paragraph, represented as a sequence of lines, each of which is a sequence of words, into one where each line has as many words as possible, not exceeding a limit of L characters. Word splitting is not allowed, and so it is assumed that any one word fits in a line. The problem is most often seen in the word-wrapping mechanisms of text editors.

This problem is typically solved with a "flow-based programming" (FBP) approach, connecting two components: a 'decomposer' and a 'recomposer'. The decomposer produces a sequence of words from the whole paragraph, and the recomposer does the harder job of building, line by line, the new paragraph.

TYPE DIAGRAM Let W(word) be a basic type and A^+ denote a non-empty list of containing elements of type A in the following type diagram:



SPECIFICATION While some find it "surprisingly difficult" to specify the telegram problem (Mateti, 2013), the relational framework used here is well equipped to express the ideas represented by the decomposer and recomposer components in the FBP perspective.

$$Telegram = Fits\ L \cdot \frac{concat}{concat} \upharpoonright (\geqslant)_{lineLength^*} \tag{A.1}$$

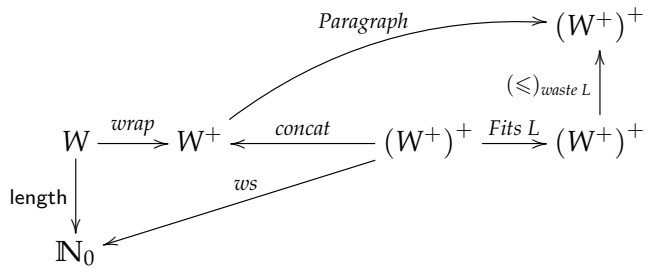
In this (post-conditioned) shrinking metaphorism, the decomposer component is the numerator of the metaphor, $\frac{concat}{}$, and the rest of the relation expression specifies the mechanism of the recomposer: build a sequence of lines ($\frac{}{concat}$), each respecting a length limit (the coreflexive relation $Fits\ L$), and make sure each line has the maximum amount of words allowed, which is achieved using a lexicographic ordering on the length of each line ($(\geqslant)_{lineLength^*}$).

A.12 THE PARAGRAPH PROBLEM

DESCRIPTION Similar to the telegram problem, but more general in nature, the paragraph problem consists of partitioning a sequence of words into different lines of a paragraph in a way which minimizes wasted space, for some definition of waste. A valid paragraph is one such that no line is longer than a given maximum length L.

More concretely, a paragraph is a non-empty sequence of lines, and a line is a non-empty sequence of words. The length of a line can be calculated by summing the length of each word, taking into account some measure of interword spaces. The function which calculates the wasted space of a paragraph is some measure of the amount of white space of each line ($ws\ l = L - \texttt{length}\ l$), with the exception of the last one, for example, the square sum.

TYPE DIAGRAM Let W(word) be a basic type and A^+ denote a non-empty list of containing elements of type A in the following type diagram:



SPECIFICATION The problem can be relationally specified as

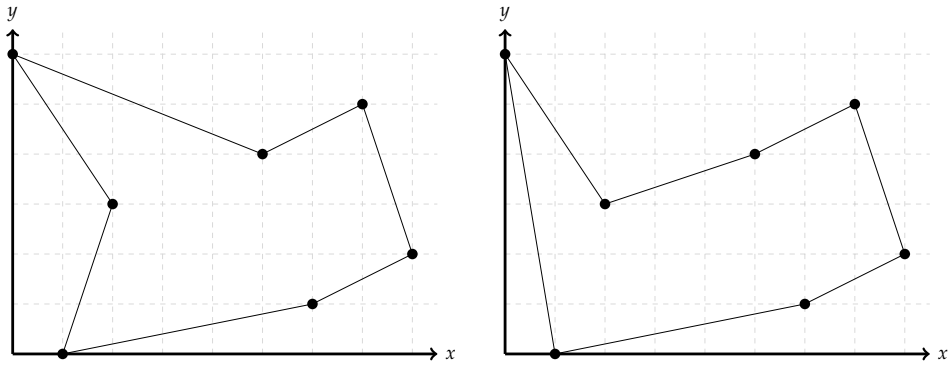
$$Paragraph = (Fits\ L) \cdot concat^{\circ} \upharpoonright (\leqslant)_{(waste\ L)} \tag{A.2}$$

where $concat^{\circ}$, the converse of the function which joins a sequences of lines, generates all possible sequences of lines, $Fits\ L$ restricts the search to those that constitute a valid paragraph, and $(\leqslant)_{(waste\ L)}$ serves to optimize for least wasted space in a paragraph.

A.13 BITONIC TOURS

DESCRIPTION The following problem is taken from (Cormen et al., 2009):

> In the *Euclidean traveling-salesman problem*, we are given a set of n points in the plane, and we wish to find the shortest closed tour that connects all n points. Figure 3a shows the solution to a 7-point problem. The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time.
>
> J.L. Bentley has suggested that we simplify the problem by restricting our attention to bitonic tours, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go



(a) Optimal tour (b) Optimal bitonic tour

Figure 3.: An optimal and an optimal bitonic tour

strictly leftward back to the starting point. Figure 3b shows the shortest bitonic tour of the same 7 points. In this case, a polynomial-time algorithm is possible.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.)

Bird and de Moor (1997) solve a generalization of this problem, where distances are not necessarily Euclidean nor necessarily symmetric. To that effect, tours are represented as a pair of lists, one for the left-to-right trip, and the other for the right-to-left trip. A valid tour must have the same first and last elements in both lists, that is, the traveller must return to its initial location, and no city should be visited twice.

The specification for this problem is simply $Tour \upharpoonright (\leqslant)_{cost}$, with $Tour$ being a relational fold that generates all possible valid tours, and $(\leqslant)_{cost}$ the optimization relation which compares the cost between tours.

A.14 THE STRING EDIT PROBLEM

DESCRIPTION The following problem is taken from (Bird and de Moor, 1997):

In the string edit problem two strings x and y are given, and it is required to transform one string into the other by performing a sequence of editing operations. There are many possible choices for these operations, but for simplicity we assume that we are given just three: $copy$, $delete$ and $insert$. Their meanings are as follows:

- $copy\ a$ copy character a from x to y;
- $delete\ a$ delete character a from x;
- $insert\ a$ insert character a in y.

The point about these operations is that if we swap the roles of $delete$ and $insert$, then we obtain a sequence transforming the target string back into the source. In fact, the operations contain enough

information to construct both strings from scratch: we merely have to interpret $copy\ a$ as meaning "append a to both strings"; $delete\ a$ as "append a to the left string"; and $insert\ a$ as "append a to the right string". Since there are many different edit sequences from which the two strings can be reconstituted, we ask for a *shortest* edit sequence.

SPECIFICATION We can specify the problem by relying on the converse of the function $edit$, which takes a sequence of operations and produces both strings: $edit^\circ \upharpoonright (\leqslant)_{\mathsf{length}}$, with $(\leqslant)_{\mathsf{length}}$ comparing the number of operation needed to transform the strings.

A.15 OPTIMAL BRACKETING

DESCRIPTION The optimal bracketing problem is a general problem that may be used to solve other more specific ones. The formulation, which is presented here, is usually instantiated for the more specific problem. A common example is the problem of building a binary tree of minimum height out of a sequence of binary trees.

Consider an expression $a_1 \oplus a_2 \oplus ... \oplus a_n$, in which $\oplus$ is an associative operation. Although all bracketings of the expression will yield the same result, different bracketings may incur different costs. In the minimum height tree problem, this means that the resulting tree will have the same elements in the same order, but is not guaranteed to have the same height.

A bracketing can be expressed as a binary leaf tree, with the values to be processed in the leaves, and the nodes indicating which values to group together. We can generate all possible bracketings through the converse of the function $flatten$, which constructs the original sequence of values from a bracketing.

SPECIFICATION The optimal bracketing problem then consists of finding the bracketing of least cost, which can be specified as $flatten^\circ \upharpoonright (\leqslant)_{cost}$.

A.16 DATA COMPRESSION

DESCRIPTION Compressing data, represented as a string of characters, can be done in multiple ways. Bird and de Moor (1997) tackle the problem of compressing a string of characters into a sequence of codes, which can be either a single character, or a pointer to a part of the string already processed. The goal is to generate the smallest possible string.

SPECIFICATION Following a similar strategy to previous examples, the problem is specified by the converse of a function $decode$, which produce the original string from the compressed one: $decode^\circ \upharpoonright (\leqslant)_{size}$, where $size$ calculates the amount of space needed to store the compressed string.

A.17 THE DETAB-ENTAB PROBLEM

DESCRIPTION The following two exercises are taken from (Kernighan and Ritchie, 1988):

> EXERCISE 1-20. Write a program *detab* that replaces tabs in the input with the proper number of blanks to space to the next tab stop. Assume a fixed set of tab stops, say every n columns. Should n be a variable or a symbolic parameter?
>
> EXERCISE 1-21. Write a program *entab* that replaces strings of blanks by the minimum number of tabs and blanks to achieve the same spacing. Use the same tab stops as for *detab*. When either a tab or a single blank would suffice to reach a tab stop, which should be given preference?

SPECIFICATION Bird and de Moor (1997) define $detab$ as a catamorphism over snoc-lists, and $entab$ is then specified as an optimum converse to $detab$, that is, $entab \subseteq detab^\circ \upharpoonright (\leqslant)_{\mathsf{length}}$, where $(\leqslant)_{\mathsf{length}}$ optimizes for number of tabs and blanks used.

A.18 THE MINIMUM TARDINESS PROBLEM

DESCRIPTION The minimum tardiness problem is a scheduling problem from operations research consisting in finding an order of execution for the given set of jobs that minimizes the maximum penalty incurred for jobs not being completed on time.

Representing the set of jobs as a list, the problem is specified by $Perm \upharpoonright (\leqslant)_{cost}$, where $Perm = \frac{bag}{bag}$ generates all possible schedules, and $(\leqslant)_{cost}$ relates two schedules by their cost, calculated by taking the maximum of the penalties for each job, considering the previously completed jobs, the time it takes to complete it, its due time, and the importance of completing that job, represented by a weight.

A.19 THE TEX PROBLEM

DESCRIPTION In the third chapter of their book, Bird and de Moor (1997) define the TEXproblem, which consists in converting from a decimal representation of a rational number to the internal TEXrepresentation of it — the closest integer multiple of 2^{-16}. In their chapter on greedy algorithms, the converse problem is tackled.

As such, constructing the *shortest* decimal representation of an integer multiple of 2^{-16} is specified as $intern^\circ \upharpoonright (\leqslant)_{\mathsf{length}}$, where intern is the aforementioned conversion function used by the TEXsystem.

B

DETAILS OF RESULTS

B.1 BACKGROUND

The proof of (B.1)

$$R \,/\, \phi_p = R \cup \frac{p}{false}$$

proceeds by ping-pong. Ping:

$$\begin{array}{ll}
& R \,/\, \phi_p \;\subseteq\; R \cup \dfrac{p}{false} \\
\equiv & \quad \{\ (2.15)\ \} \\
& R \,/\, \phi_p - \dfrac{p}{false} \;\subseteq\; R \\
\equiv & \quad \{\ \text{relational difference (2.16)}; \neg \tfrac{p}{false} = \tfrac{p}{true}\ (2.184)\ \} \\
& R \,/\, \phi_p \cap \dfrac{p}{true} \;\subseteq\; R \\
\equiv & \quad \{\ R \text{ is a connected preorder, so surjective and entire, apply (2.166)}\ \} \\
& (R \,/\, \phi_p) \cdot \phi_p \;\subseteq\; R \\
\equiv & \quad \{\ \text{right division (2.108)}\ \} \\
& R \,/\, \phi_p \;\subseteq\; R \,/\, \phi_p \\
\Box &
\end{array}$$

Pong:

$$\begin{array}{ll}
& R \cup \dfrac{p}{false} \;\subseteq\; R \,/\, \phi_p \\
\equiv & \quad \{\ \text{universal proporty of meet (2.13)}\ \} \\
& \left\{ \begin{array}{l} R \cdot \phi_p \;\subseteq\; R \\ \frac{p}{false} \;\subseteq\; R \,/\, \phi_p \end{array} \right.
\end{array}$$

$$
\begin{array}{ll}
\equiv & \{\ \phi_p \subseteq id \text{ (2.160), monotonicity (2.32); right division (2.108)}\ \} \\
& \frac{p}{false} \cdot \phi_p \ \subseteq\ R \\
\Leftarrow & \{\ \text{monotonicity (2.32)}\ \} \\
& \frac{p}{false} \cdot \frac{true}{p} \ \subseteq\ R \\
\equiv & \{\ \text{symmetric division (2.115); } \frac{true}{false} = \bot\ \} \\
& \bot \ \subseteq\ R \\
\Box &
\end{array}
$$

The proof of (2.166)

$$
\begin{array}{ll}
& R \cdot \phi_p \\
= & \{\ \text{definition of } \phi_p \text{ (2.159)}\ \} \\
& R \cdot (id \cap \frac{p}{true}) \\
= & \{\ \text{meet right distributivity (2.24)}\ \} \\
& R \cap R \cdot \frac{p}{true} \\
= & \{\ \text{symmetric division (2.112); converse}\ \} \\
& R \cap (true \cdot R^\circ)^\circ \cdot p \\
= & \{\ R^\circ \text{ is entire and } true \text{ is constant (2.180); symmetric division (2.112)}\ \} \\
& R \cap \frac{true}{p} \\
\Box &
\end{array}
$$

Side condition for right distributivity:

$$\ker R \cdot \frac{p}{true} \subseteq \frac{p}{true}$$

Proof:

$$
\begin{array}{ll}
& \ker R \cdot \frac{p}{true} \subseteq \frac{p}{true} \\
= & \{\ \text{definition of } \ker \text{ (2.38); symmetric division (2.112); converse}\ \} \\
& (true \cdot R^\circ \cdot R)^\circ \cdot p \subseteq true^\circ \cdot p \\
= & \{\ R \text{ is surjective and entire, and } true \text{ is constant, apply (2.180) twice; symmetric division (2.112)}\ \} \\
& \frac{p}{true} \subseteq \frac{p}{true} \\
\Box &
\end{array}
$$

Exercise 7.38 from Bird and de Moor (1997). For S monotonic with respect to R°, where R is a preorder:

$$
\begin{array}{ll}
& (S \upharpoonright R) \cdot (\in \upharpoonright \mathbb{F}\ R) \subseteq S \cdot \in \upharpoonright R \\
\equiv & \quad \{ \text{ universal property of shrinking (2.185); converse } \} \\
& \left\{ \begin{array}{l} (S \upharpoonright R) \cdot (\in \upharpoonright \mathbb{F}\ R) \subseteq S \cdot \in \\ (S \upharpoonright R) \cdot (\in \upharpoonright \mathbb{F}\ R) \cdot (S \cdot \in)^\circ \subseteq R \end{array} \right. \\
\Leftarrow & \quad \{ \text{ shrinking cancellation (2.187) (twice) } \} \\
& \left\{ \begin{array}{l} S \cdot \in \subseteq S \cdot \in \\ (S \upharpoonright R) \cdot (\in \upharpoonright \mathbb{F}\ R) \cdot \in^\circ \cdot S^\circ \subseteq R \end{array} \right. \\
\Leftarrow & \quad \{ \text{ shrinking cancellation (2.188) } \} \\
& (S \upharpoonright R) \cdot \mathbb{F}\ R \cdot S^\circ \subseteq R \\
\Leftarrow & \quad \{ \ S \text{ is monotonic with respect to } R^\circ \ \} \\
& (S \upharpoonright R) \cdot S^\circ \cdot R \subseteq R \\
\Leftarrow & \quad \{ \text{ shrinking cancellation (2.188) } \} \\
& R \cdot R \subseteq R \\
\equiv & \quad \{ \ R \text{ is transitive (2.37) } \} \\
& true \\
\Box &
\end{array}
$$

Proof of (2.202):

$$
(S + T) \upharpoonright (R + Q) = S \upharpoonright R + T \upharpoonright Q
$$

$$
\begin{array}{ll}
& X \subseteq (S + T) \upharpoonright (R + Q) \\
\equiv & \quad \{ \text{ universal law of shrinking (2.185) } \} \\
& \left\{ \begin{array}{l} X \subseteq S + T \\ X \cdot (S + T)^\circ \subseteq R + Q \end{array} \right. \\
\equiv & \quad \{ \ X := X_1 + X_2 \text{; converse } \} \\
& \left\{ \begin{array}{l} X_1 + X_2 \subseteq S + T \\ (X_1 + X_2) \cdot (S^\circ + T^\circ) \subseteq R + Q \end{array} \right. \\
\equiv & \quad \{ \text{ direct sum fusion (2.88); direct sum inequality (2.89) } \}
\end{array}
$$

$$
\begin{cases} X_1 \subseteq S \\ X_2 \subseteq T \\ X_1 \cdot S^\circ \subseteq R \\ X_2 \cdot T^\circ \subseteq Q \end{cases}
$$

$\equiv$ { universal law of shrinking (2.185) (twice) }

$$
\begin{cases} X_1 \subseteq S \upharpoonright R \\ X_2 \subseteq T \upharpoonright Q \end{cases}
$$

$\equiv$ { direct sum inequality (2.89); $X := X_1 + X_2$ }

$$
X \subseteq S \upharpoonright R + T \upharpoonright Q
$$

:: { indirect equality (2.9) }

$$
(S + T) \upharpoonright (R + Q) = S \upharpoonright R + T \upharpoonright Q
$$

□

Generalization of exercise 7.15 from (Bird and de Moor, 1997):

$$
\langle S \upharpoonright R, T \upharpoonright Q \rangle \subseteq \langle S, T \rangle \upharpoonright (R \times Q)
$$

$\equiv$ { universal property of shrinking (2.185) }

$$
\begin{cases} \langle S \upharpoonright R, T \upharpoonright Q \rangle \subseteq \langle S, T \rangle \\ \langle S \upharpoonright R, T \upharpoonright Q \rangle \cdot \langle S, T \rangle^\circ \subseteq R \times Q \end{cases}
$$

$\equiv$ { pairings (2.60, 2.62, 2.63), Kronecker product definition (2.70) }

$$
\begin{cases} S \upharpoonright R \subseteq S \\ T \upharpoonright Q \subseteq T \\ \pi_1 \cdot \langle S \upharpoonright R, T \upharpoonright Q \rangle \cdot \langle S, T \rangle^\circ \subseteq R \cdot \pi_1 \\ \pi_2 \cdot \langle S \upharpoonright R, T \upharpoonright Q \rangle \cdot \langle S, T \rangle^\circ \subseteq Q \cdot \pi_2 \end{cases}
$$

$\Leftarrow$ { shrinking cancellation (2.187) (twice); pairing cancellation (2.62, 2.63) }

$$
\begin{cases} (S \upharpoonright R) \cdot \langle S, T \rangle^\circ \subseteq R \cdot \pi_1 \\ (T \upharpoonright Q) \cdot \langle S, T \rangle^\circ \subseteq Q \cdot \pi_2 \end{cases}
$$

$\equiv$ { shunting (2.59); converse }

$$
\begin{cases} (S \upharpoonright R) \cdot (\pi_1 \cdot \langle S, T \rangle)^\circ \subseteq R \\ (T \upharpoonright Q) \cdot (\pi_2 \cdot \langle S, T \rangle)^\circ \subseteq Q \end{cases}
$$

$\Leftarrow$ { pairing cancellation (2.62, 2.63) }

$$
\begin{cases} (S \upharpoonright R) \cdot S^\circ \subseteq R \\ (T \upharpoonright Q) \cdot T^\circ \subseteq Q \end{cases}
$$

$\equiv$ { shrinking cancellation (2.188) (twice) }

$$true$$

□

B.2 INDUCTIVE RELATIONS ON LISTS

For equality (3.38),

$$Select \cdot (\sqsubseteq) = (id \times (\sqsubseteq)) \cdot Select$$

(postulated in page 56) to be proved, one needs to inspect (higher order) catamorphism $bag : \mathbb{Z}^A \leftarrow A^*$:

$$\begin{cases} bag\ [\,]\ a = 0 \\ bag\ (h : t)\ a = bag\ t\ a + (\mathbf{if}\ a = h\ \mathbf{then}\ 1\ \mathbf{else}\ 0) \end{cases} \tag{B.1}$$

We choose $\mathbb{Z}$ and not $\mathbb{N}_0$ in the type of bag to endow it with vector space operations, e.g. addition (also subtraction etc)

$$(f + g)\ a = (f\ a) + (g\ a) \tag{B.2}$$

where A generates the basis, made of "vectors"

$$\hat{a}\ b = \mathbf{if}\ a = b\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \tag{B.3}$$

etc. In this setting, $bag = (\!|\, [\underline{0}, g] \,|\!)$ where $g\ (a, f) = \hat{a} + f$. By unfolding the definition of $Select$ one gets:

$$(a, y)\ Select\ x \Leftrightarrow (a : y)\ Perm\ x \Leftrightarrow bag\ x = \hat{a} + bag\ y.$$

Going pointwise on this,

$$\begin{array}{ccccccccc} & & Select & \cdot & (\sqsubseteq) & = & & (id \times (\sqsubseteq)) & \cdot & Select \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \vdots \\ (a, y) & & z & & x & & (a, y) & & (a, y') & x \end{array}$$

(3.38) states that, for all a,x and y, there exist z and y' such that

$$\begin{cases} (a, y')\ Select\ x \\ y \sqsubseteq y' \end{cases} \equiv \begin{cases} z \sqsubseteq x \\ (a, y)\ Select\ z \end{cases}$$

$$\equiv \quad \{\ \text{unfold } Select\ \}$$

$$\begin{cases} \hat{a} + bag\ y' = bag\ x \\ y \sqsubseteq y' \end{cases} \equiv \begin{cases} \hat{a} + bag\ y = bag\ z \\ z \sqsubseteq x \end{cases}$$

This holds provided subsequence z of x and supersequence y' of y are chosen such that $bag\ x - bag\ y' = bag\ z - bag\ y$. This condition can always be met. Given z, a suitable y' can be chosen as a function of z, and vice-versa, because the resulting bags are guaranteed not to contain any negative multiplicities.

To prove (3.39),

$$(\sqsubseteq) \cdot Discard \subseteq Perm \cdot (\sqsubseteq)$$

we take a similar approach:

$$\begin{array}{ccccccccc} (\sqsubseteq) & \cdot & Discard & \subseteq & Perm & \cdot & (\sqsubseteq) \\ \vdots & & \vdots & & \vdots & & \vdots \\ y & & z & & x & y & y' & & x \end{array}$$

(3.39) states that, for all x, $a \in x$ and y, there exist z and y' such that

$$\begin{array}{ll} & \left\{ \begin{array}{l} z\ Discard\ x \\ y \sqsubseteq z \end{array} \right. \Rightarrow \left\{ \begin{array}{l} y' \sqsubseteq x \\ y\ Perm\ y' \end{array} \right. \\ \equiv & \quad \{ \text{ unfold } Discard \text{ and } Perm \ \} \\ & \left\{ \begin{array}{l} bag\ z = bag\ x - \hat{a} \\ y \sqsubseteq z \end{array} \right. \Rightarrow \left\{ \begin{array}{l} y' \sqsubseteq x \\ bag\ y = bag\ y' \end{array} \right. \end{array}$$

Choose any y' that results from removing from x all the elements that were removed from z to form y, and a. This guarantees that $bag\ y = bag\ y'$ and can be done such that $y' \sqsubseteq x$.

Proof of (3.31):

$$\begin{array}{ll} & \epsilon \cdot Perm = \epsilon \\ \equiv & \quad \{ \text{ pong step only: } Perm \text{ is reflexive (2.35) } \} \\ & \epsilon \cdot Perm \subseteq \epsilon \\ \equiv & \quad \{ \text{ in-out isomorphism } \} \\ & \epsilon \cdot \mathsf{in} \cdot \mathsf{out} \cdot Perm \subseteq \epsilon \\ \equiv & \quad \{ \text{ (2.224); converse; catamorphism (2.204) } \} \\ & [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \cdot \mathbb{F}\ Perm \cdot [nil, Select^\circ]^\circ \subseteq \epsilon \\ \equiv & \quad \{ \text{ coproducts (2.90); biproduct absorption (2.85) } \} \\ & \left\{ \begin{array}{l} \bot \cdot nil^\circ \subseteq \epsilon \\ (\pi_1 \cup \epsilon \cdot \pi_2) \cdot (id \times Perm) \cdot Select \subseteq \epsilon \end{array} \right. \\ \equiv & \quad \{ \text{ shunting (2.59); } \epsilon \cdot nil = \bot \text{ (3.2); } Select = (id \times Perm) \cdot Select \text{ (3.23) } \} \\ & (\pi_1 \cup \epsilon \cdot \pi_2) \cdot Select \subseteq \epsilon \\ \equiv & \quad \{ \text{ left linearity of join (2.21); universal property of join (2.14) } \} \\ & \left\{ \begin{array}{l} \pi_1 \cdot Select \subseteq \epsilon \\ \epsilon \cdot \pi_2 \cdot Select \subseteq \epsilon \end{array} \right. \end{array}$$

$$
\begin{array}{ll}
\equiv & \{ \text{ properties of } \mathit{Select} \text{ (3.25, 3.26) } \} \\
& \mathit{true} \\
\Box &
\end{array}
$$

Proof of (3.34):

$$
\begin{array}{ll}
& \phi_p \cdot \mathit{Perm} \\
= & \{ \text{ definition of } \phi_p \text{ (2.159) } \} \\
& (\mathit{id} \cap \dfrac{\mathit{true}}{p}) \cdot \mathit{Perm} \\
= & \{ \text{ meet distributivity (2.25), side condition } \frac{\mathit{true}}{p} \cdot \mathrm{img}\ \mathit{Perm} \subseteq \frac{\mathit{true}}{p} \text{ holds } \} \\
& \mathit{Perm} \cap \dfrac{\mathit{true}}{p} \cdot \mathit{Perm} \\
= & \{ \text{ symmetric division (2.112) } \} \\
& \mathit{Perm} \cap p^\circ \cdot \mathit{true} \cdot \mathit{Perm} \\
= & \{ \ p \text{ and } \mathit{true} \text{ are stable (3.32) } \} \\
& \mathit{Perm} \cap (p \cdot \mathit{Perm})^\circ \cdot \mathit{true} \\
= & \{ \text{ converse; } \mathit{Perm} = \mathit{Perm}^\circ \text{; symmetric division (2.112) } \} \\
& \mathit{Perm} \cap \mathit{Perm} \cdot \dfrac{\mathit{true}}{p} \\
= & \{ \text{ meet distributivity (2.24), side condition } \ker \mathit{Perm} \cdot \frac{\mathit{true}}{p} \subseteq \frac{\mathit{true}}{p} \text{ holds } \} \\
& \mathit{Perm} \cdot (\mathit{id} \cap \dfrac{\mathit{true}}{p}) \\
= & \{ \text{ definition of } \phi_p \text{ (2.159) } \} \\
& \mathit{Perm} \cdot \phi_p \\
\Box &
\end{array}
$$

Side condition for left distributivity:

$$\frac{\mathit{true}}{p} \cdot \mathrm{img}\ \mathit{Perm} \subseteq \frac{\mathit{true}}{p}$$

Proof:

$$
\begin{array}{ll}
& \dfrac{\mathit{true}}{p} \cdot \mathrm{img}\ \mathit{Perm} \\
= & \{ \text{ symmetric division (2.112); } \mathit{Perm} = \frac{\mathit{bag}}{\mathit{bag}}, \text{ so } \mathrm{img}\ \mathit{Perm} = \mathit{Perm} \text{ (2.120) } \} \\
& p^\circ \cdot \mathit{true} \cdot \mathit{Perm} \\
= & \{ \ \mathit{Perm} \text{ is entire, } \mathit{true} \text{ is a constant function (2.180) } \}
\end{array}
$$

$$
\begin{array}{ll}
& p^\circ \cdot true \\
= & \quad \{ \text{ symmetric division (2.112) } \} \\
& \overline{\frac{true}{p}} \\
\square &
\end{array}
$$

Side condition for right distributivity:

$$\ker Perm \cdot \frac{true}{p} \subseteq \frac{true}{p}$$

Proof:

$$
\begin{array}{ll}
& \ker Perm \cdot \frac{true}{p} \\
= & \quad \{ \text{ symmetric division (2.112); } Perm = \frac{bag}{bag}, \text{ so } \ker Perm = Perm \text{ (2.119) } \} \\
& Perm \cdot p^\circ \cdot true \\
= & \quad \{ \text{ converse; } Perm = Perm^\circ \text{ (3.13); } p \text{ is stable (3.32) } \} \\
& p^\circ \cdot true \\
= & \quad \{ \text{ symmetric division (2.112) } \} \\
& \overline{\frac{true}{p}} \\
\square &
\end{array}
$$

Proof of (3.37):

$$
\begin{array}{ll}
& \epsilon \cdot (\sqsubseteq) \subseteq \epsilon \\
\equiv & \quad \{ \text{ left division (2.91) } \} \\
& (\sqsubseteq) \subseteq \epsilon \setminus \epsilon \\
\Leftarrow & \quad \{ \text{ cata fusion (2.215) } \} \\
& [nil, cons \cup \pi_2] \cdot \mathbb{F}\ (\epsilon \setminus \epsilon) \subseteq (\epsilon \setminus \epsilon) \cdot \mathsf{in} \\
\equiv & \quad \{ \text{ (2.101); left division (2.91) } \} \\
& \epsilon \cdot [nil, cons \cup \pi_2] \cdot \mathbb{F}\ (\epsilon \setminus \epsilon) \subseteq \epsilon \cdot \mathsf{in} \\
\equiv & \quad \{ \ \epsilon \cdot \mathsf{in} = [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \text{ (2.224) } \} \\
& \epsilon \cdot [nil, cons \cup \pi_2] \cdot \mathbb{F}\ (\epsilon \setminus \epsilon) \subseteq [\bot, \pi_1 \cup \epsilon \cdot \pi_2] \\
\equiv & \quad \{ \text{ coproducts (2.90); } \epsilon \cdot nil = \bot \text{ (3.2) } \} \\
& \epsilon \cdot (cons \cup \pi_2) \cdot (id \times (\epsilon \setminus \epsilon)) \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\equiv & \quad \{ \text{ bilinearity of join (2.20, 2.21) } \} \\
& \epsilon \cdot cons \cdot (id \times (\epsilon \setminus \epsilon)) \cup \epsilon \cdot \pi_2 \cdot (id \times (\epsilon \setminus \epsilon)) \subseteq \pi_1 \cup \epsilon \cdot \pi_2
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{ \text{ Kronecker product cancellation (2.75) } \} \\
& \epsilon \cdot cons \cdot (id \times (\epsilon \setminus \epsilon)) \cup \epsilon \cdot (\epsilon \setminus \epsilon) \cdot \pi_2 \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\Leftarrow & \{ \text{ division cancellation (2.93) } \} \\
& \epsilon \cdot cons \cdot (id \times (\epsilon \setminus \epsilon)) \cup \epsilon \cdot \pi_2 \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\equiv & \{ \text{ universal property of join (2.14) means } \epsilon \cdot \pi_2 \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \text{ } \} \\
& \epsilon \cdot cons \cdot (id \times (\epsilon \setminus \epsilon)) \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\equiv & \{ \epsilon \cdot cons = \pi_1 \cup \epsilon \cdot \pi_2 \text{ (3.3) } \} \\
& (\pi_1 \cup \epsilon \cdot \pi_2) \cdot (id \times (\epsilon \setminus \epsilon)) \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\equiv & \{ \text{ bilinearity of join (2.21); Kronecker product cancellation (2.74, 2.75) } \} \\
& \pi_1 \cup \epsilon \cdot (\epsilon \setminus \epsilon) \cdot \pi_2 \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\Leftarrow & \{ \text{ division cancellation (2.93) } \} \\
& \pi_1 \cup \epsilon \cdot \pi_2 \subseteq \pi_1 \cup \epsilon \cdot \pi_2 \\
\Box &
\end{array}
$$

The proof of (3.47) proceeds by ping-pong:

$$
\begin{array}{ll}
& (\sqsubseteq) \cdot Perm \subseteq Perm \cdot (\sqsubseteq) \\
\Leftarrow & \{ \text{ hylomorphism least fixpoint (2.220) } \} \\
& [nil, cons \cup \pi_2] \cdot \mathbb{F}\ (Perm \cdot (\sqsubseteq)) \cdot [nil, Select^\circ]^\circ \subseteq Perm \cdot (\sqsubseteq) \\
\equiv & \{ \text{ direct sum and biproduct absorption (2.87, 2.85); universal property of join (2.14) } \} \\
& \left\{ \begin{array}{l} nil \cdot nil^\circ \subseteq Perm \cdot (\sqsubseteq) \\ (cons \cup \pi_2) \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \subseteq Perm \cdot (\sqsubseteq) \end{array} \right. \\
\equiv & \{ \text{ } nil \text{ is simple (2.41); } Perm \text{ and } (\sqsubseteq) \text{ and reflexive (2.35) } \} \\
& (cons \cup \pi_2) \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \subseteq Perm \cdot (\sqsubseteq) \\
\equiv & \{ \text{ join bilinearity (2.21); universal property of join (2.14) } \} \\
& \left\{ \begin{array}{l} cons \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \subseteq Perm \cdot (\sqsubseteq) \\ \pi_2 \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \subseteq Perm \cdot (\sqsubseteq) \end{array} \right.
\end{array}
$$

The first condition is proven by exploiting the interaction between $Select$ and $cons$:

$$
\begin{array}{ll}
& cons \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \subseteq Perm \cdot (\sqsubseteq) \\
\equiv & \{ \text{ } Select \cdot (\sqsubseteq) = (id \times (\sqsubseteq)) \cdot Select \text{ (3.38) } \} \\
& cons \cdot (id \times Perm) \cdot Select \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \\
\equiv & \{ \text{ } Select \cdot Perm = (id \times Perm) \cdot Select \text{ (3.23) } \} \\
& cons \cdot Select \cdot Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq)
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \{\ cons \cdot Select \subseteq Perm\ (3.21)\ \} \\
& Perm \cdot Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \\
\Leftarrow & \{\ Perm \text{ is transitive } (2.37)\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \\
\square &
\end{array}
$$

The second condition rests on (3.39) which involves the $Discard$ relation:

$$
\begin{array}{ll}
& \pi_2 \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \subseteq Perm \cdot (\sqsubseteq) \\
\equiv & \{\ \text{Kronecker product cancellation } (2.75); \text{ definition of } Discard\ \} \\
& Perm \cdot (\sqsubseteq) \cdot Discard \subseteq Perm \cdot (\sqsubseteq) \\
\equiv & \{\ (3.39)\ \} \\
& Perm \cdot Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \\
\Leftarrow & \{\ Perm \text{ is transitive } (2.37)\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \\
\square &
\end{array}
$$

The pong step also uses a fixpoint law:

$$
\begin{array}{ll}
& Perm \cdot (\sqsubseteq) \subseteq (\sqsubseteq) \cdot Perm \\
\Leftarrow & \{\ \text{hylomorphism least fixpoint } (2.221)\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq [nil, cons \cup \pi_2] \cdot \mathbb{F}\ (Perm \cdot (\sqsubseteq)) \cdot [nil, Select^\circ]^\circ \\
\equiv & \{\ \text{direct sum and biproduct absorption } (2.87, 2.85)\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq nil \cdot nil^\circ \cup (cons \cup \pi_2) \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \\
\Leftarrow & \{\ \text{lower the upper side } (2.31) \text{ with } S \subseteq S \cup R\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq nil \cdot nil^\circ \cup cons \cdot (id \times Perm \cdot (\sqsubseteq)) \cdot Select \\
\equiv & \{\ \text{from } (3.38) \text{ and } (3.23), \text{ we have that } Select \cdot Perm \cdot (\sqsubseteq) = (id \times Perm \cdot (\sqsubseteq)) \cdot Select\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq nil \cdot nil^\circ \cup cons \cdot Select \cdot Perm \cdot (\sqsubseteq) \\
\Leftarrow & \{\ \phi_{\neg\, null} \subseteq cons \cdot Select\ (3.22); \phi_{null} = nil \cdot nil^\circ\ (3.9)\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq \phi_{null} \cup \phi_{\neg\, null} \cdot Perm \cdot (\sqsubseteq) \\
\Leftarrow & \{\ \text{claim: } \phi_{\neg\, null} \cdot Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \cdot \phi_{\neg\, null}\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq \phi_{null} \cup Perm \cdot (\sqsubseteq) \cdot \phi_{\neg\, null} \\
\equiv & \{\ \text{biproduct absorption } (2.85); \text{ definition of guard } (2.169)\ \} \\
& Perm \cdot (\sqsubseteq) \subseteq [id, Perm \cdot (\sqsubseteq)] \cdot null? \\
\equiv & \{\ \text{shunting } (2.59)\ \}
\end{array}
$$

$$
\begin{array}{ll}
& Perm \cdot (\sqsubseteq) \cdot [\phi_{null}, \phi_{\neg\, null}] \subseteq [id, Perm \cdot (\sqsubseteq)] \\
\equiv & \quad \{ \text{ coproducts (2.90) } \} \\
& \left\{ \begin{array}{l} Perm \cdot (\sqsubseteq) \cdot \phi_{null} \subseteq id \\ Perm \cdot (\sqsubseteq) \cdot \phi_{\neg\, null} \subseteq Perm \cdot (\sqsubseteq) \end{array} \right. \\
\Leftarrow & \quad \{ \text{ raise the lower side (2.32) with } \phi_p \subseteq id \text{ due to (2.160) } \} \\
& Perm \cdot (\sqsubseteq) \cdot \phi_{null} \subseteq id \\
\equiv & \quad \{ \text{ claim: } Perm \cdot (\sqsubseteq) \cdot \phi_{null} = \phi_{null} \ \} \\
& \phi_{null} \subseteq id \\
\equiv & \quad \{ \ \phi_p \subseteq id \text{ (2.160) } \} \\
& true \\
\Box &
\end{array}
$$

The claims pertain to the emptiness of a list in relation to a sublist, and are easy to prove given the similar results for $(\sqsubseteq)$ and $Perm$.

$$\phi_{\neg\, null} \cdot Perm \cdot (\sqsubseteq) \subseteq Perm \cdot (\sqsubseteq) \cdot \phi_{\neg\, null}$$

Proof:

$$
\begin{array}{ll}
& \phi_{\neg\, null} \cdot Perm \cdot (\sqsubseteq) \\
= & \quad \{ \ \neg\, null \text{ is stable, so } \phi_{\neg\, null} \cdot Perm = Perm \cdot \phi_{\neg\, null} \text{ (3.34) } \} \\
& Perm \cdot \phi_{\neg\, null} \cdot (\sqsubseteq) \\
\subseteq & \quad \{ \text{ (3.41) } \} \\
& Perm \cdot (\sqsubseteq) \cdot \phi_{\neg\, null} \\
\Box &
\end{array}
$$

$$Perm \cdot (\sqsubseteq) \cdot \phi_{null} = \phi_{null}$$

Proof:

$$
\begin{array}{ll}
& Perm \cdot (\sqsubseteq) \cdot \phi_{null} \\
= & \quad \{ \text{ (3.40) } \} \\
& Perm \cdot \phi_{null} \\
= & \quad \{ \text{ (3.16) } \} \\
& \phi_{null}
\end{array}
$$

□

Proof of (3.49):

$$
\begin{array}{ll}
& R_{f^*} \xleftarrow{cons} \mathbb{G}\ R_{f^*} \\
\equiv & \quad \{\ \text{definition}\ \} \\
& cons \cdot \mathbb{G}\ R_{f^*} \subseteq R_{f^*} \cdot cons \\
\equiv & \quad \{\ \text{abbreviation (2.6); shunting (2.58); converse}\ \} \\
& \mathbb{G}\ (f^*)^\circ \cdot \mathbb{G}\ R \cdot \mathbb{G}\ f^* \subseteq (f^* \cdot cons)^\circ \cdot R \cdot f^* \cdot cons \\
\Leftarrow & \quad \{\ \text{free theorem of } cons \text{ (3.5) (twice); converse}\ \} \\
& \mathbb{G}\ (f^*)^\circ \cdot \mathbb{G}\ R \cdot \mathbb{G}\ f^* \subseteq (f \times f^*)^\circ \cdot cons^\circ \cdot R \cdot cons \cdot (f \times f^*) \\
\equiv & \quad \{\ \text{shunting (2.58, 2.59); } \mathbb{G}\ R = id \times R\ \} \\
& (w \times f^*) \cdot (id \times (f^*)^\circ) \cdot (id \times R) \cdot (id \times f^*) \cdot (f \times f^*)^\circ \subseteq cons^\circ \cdot R \cdot cons \\
\equiv & \quad \{\ \text{converse; } \times \text{ is a bifunctor}\ \} \\
& f \cdot f^\circ \times f^* \cdot (f^*)^\circ \cdot R \cdot f^* \cdot (f^*)^\circ \subseteq cons^\circ \cdot R \cdot cons \\
\equiv & \quad \{\ f \text{ and } f^* \text{ are simple (2.41), monotonicity (2.32); definition of } \mathbb{G}\ \} \\
& \mathbb{G}\ R \subseteq cons^\circ \cdot R \cdot cons \\
\equiv & \quad \{\ \text{shunting (2.58)}\ \} \\
& cons \cdot \mathbb{G}\ R \subseteq R \cdot cons \\
\equiv & \quad \{\ \text{definition}\ \} \\
& R \xleftarrow{cons} \mathbb{G}\ R
\end{array}
$$

□

Proof of (3.50):

$$
\begin{array}{ll}
& S_{(\!|g|\!)} \xleftarrow{\mathsf{in}} \mathbb{F}\ S_{(\!|g|\!)} \\
\equiv & \quad \{\ \text{definition (2.132)}\ \} \\
& \mathsf{in} \cdot \mathbb{F}\ S_{(\!|g|\!)} \subseteq S_{(\!|g|\!)} \cdot \mathsf{in} \\
\equiv & \quad \{\ \text{abbreviation (2.6); shunting (2.58); converse}\ \} \\
& \mathbb{F}\ (\!|g|\!)^\circ \cdot \mathbb{F}\ S \cdot \mathbb{F}\ (\!|g|\!) \subseteq (\!|g|\!) \cdot \mathsf{in})^\circ \cdot S \cdot (\!|g|\!) \cdot \mathsf{in} \\
\equiv & \quad \{\ \text{cata (2.204) (twice); converse}\ \} \\
& \mathbb{F}\ (\!|g|\!)^\circ \cdot \mathbb{F}\ S \cdot \mathbb{F}\ (\!|g|\!) \subseteq \mathbb{F}\ (\!|g|\!)^\circ \cdot g^\circ \cdot S \cdot g \cdot \mathbb{F}\ (\!|g|\!) \\
\Leftarrow & \quad \{\ \text{monotonicity (2.27) (twice); shunting (2.58)}\ \} \\
& g \cdot \mathbb{F}\ S \subseteq S \cdot g
\end{array}
$$

$$\equiv \quad \{ \text{ definition (2.132) } \}$$

$$S \xleftarrow{g} \mathbb{F}\ S$$

□

B.3 THE GREEDY ALGORITHM

Proof of (4.5):

$$\begin{aligned}
& \phi_p \cdot cons = \phi_p \cdot cons \cdot (id \times \phi_p) \\
\equiv & \quad \{ \text{ with } \phi_p \subseteq id \text{ (2.160), monotonicity (2.32) leaves only } \} \\
& \phi_p \cdot cons \subseteq \phi_p \cdot cons \cdot (id \times \phi_p) \\
\equiv & \quad \{ \ \phi_p \cdot \phi_p = \phi_p \text{ (2.163); (4.4) } \} \\
& \phi_p \cdot cons \cdot (id \times \phi_p) \subseteq \phi_p \cdot cons \cdot (id \times \phi_p)
\end{aligned}$$

□

Claim: $(\!|\, [nil, \phi_p \cdot (cons \cup \pi_2)] \,|\!) = (\!|\, [nil, \phi_p \cdot cons \cup \pi_2] \,|\!)$

$$\begin{aligned}
& (\!|\, \underbrace{[nil, \phi_p \cdot (cons \cup \pi_2)]}_{S} \,|\!) = (\!|\, [nil, \phi_p \cdot cons \cup \pi_2] \,|\!) \\
\equiv & \quad \{ \text{ universal property (2.205) } \} \\
& (\!|\, [nil, \phi_p \cdot (cons \cup \pi_2)] \,|\!) \cdot \mathsf{in} = [nil, \phi_p \cdot cons \cup \pi_2] \cdot \mathbb{F}\ (\!|\, S \,|\!) \\
\equiv & \quad \{ \text{ cata cancellation (2.204) } \} \\
& [nil, \phi_p \cdot (cons \cup \pi_2)] \cdot \mathbb{F}\ (\!|\, S \,|\!) = [nil, \phi_p \cdot cons \cup \pi_2] \cdot \mathbb{F}\ (\!|\, S \,|\!) \\
\equiv & \quad \{ \text{ coproducts (2.90) } \} \\
& \begin{cases} nil = nil \\ \phi_p \cdot (cons \cup \pi_2) \cdot (id \times (\!|\, S \,|\!)) = (\phi_p \cdot cons \cup \pi_2) \cdot (id \times (\!|\, S \,|\!)) \end{cases}
\end{aligned}$$

First condition is trivial. Second condition:

$$\begin{aligned}
& \phi_p \cdot (cons \cup \pi_2) \cdot (id \times (\!|\, S \,|\!)) \\
= & \quad \{ \text{ bilinearity of join (2.21, 2.20) } \} \\
& \phi_p \cdot cons \cdot (id \times (\!|\, S \,|\!)) \cup \phi_p \cdot \pi_2 \cdot (id \times (\!|\, S \,|\!)) \\
= & \quad \{ \text{ Kronecker product cancellation (2.75) } \} \\
& \phi_p \cdot cons \cdot (id \times (\!|\, S \,|\!)) \cup \phi_p \cdot (\!|\, S \,|\!) \cdot \pi_2 \\
= & \quad \{ \text{ claim: } \phi_p \cdot (\!|\, S \,|\!) = (\!|\, S \,|\!) \ \} \\
& \phi_p \cdot cons \cdot (id \times (\!|\, S \,|\!)) \cup (\!|\, S \,|\!) \cdot \pi_2
\end{aligned}$$

$$
\begin{array}{ll}
= & \{ \text{ Kronecker product cancellation (2.75) } \} \\
& \phi_p \cdot cons \cdot (id \times (\!| S |\!)) \cup \pi_2 \cdot (id \times (\!| S |\!)) \\
= & \{ \text{ bilinearity of join (2.21) } \} \\
& (\phi_p \cdot cons \cup \pi_2) \cdot (id \times (\!| S |\!)) \\
\Box &
\end{array}
$$

Claim: $\phi_p \cdot (\!| S |\!) = (\!| S |\!)$

$$
\begin{array}{ll}
& \phi_p \cdot (\!| S |\!) = (\!| S |\!) \\
\Leftarrow & \{ \text{ cata fusion (2.213) } \} \\
& \phi_p \cdot S = S \cdot \mathbb{F}\ \phi_p \\
\equiv & \{ \ S = [\phi_p \cdot nil, \phi_p \cdot (cons \cup \pi_2)]; \text{ coproducts (2.90) } \} \\
& \left\{ \begin{array}{l} \phi_p \cdot \phi_p \cdot nil = \phi_p \cdot nil \\ \phi_p \cdot \phi_p \cdot (cons \cup \pi_2) = \phi_p \cdot (cons \cup \pi_2) \cdot (id \times \phi_p) \end{array} \right. \\
\equiv & \{ \ \phi_p \cdot \phi_p = \phi_p \text{ (2.163) } \} \\
& \phi_p \cdot (cons \cup \pi_2) = \phi_p \cdot (cons \cup \pi_2) \cdot (id \times \phi_p)
\end{array}
$$

Proceeding by direct equality:

$$
\begin{array}{ll}
& \phi_p \cdot (cons \cup \pi_2) \cdot (id \times \phi_p) \\
= & \{ \text{ bilinearity of join (2.21, 2.20) } \} \\
& \phi_p \cdot cons \cdot (id \times \phi_p) \cup \phi_p \cdot \pi_2 \cdot (id \times \phi_p) \\
= & \{ \ \phi_p \cdot cons = \phi_p \cdot cons \cdot (id \times \phi_p) \text{ (4.5) } \} \\
& \phi_p \cdot cons \cup \phi_p \cdot \pi_2 \cdot (id \times \phi_p) \\
= & \{ \text{ Kronecker product cancellation (2.75) } \} \\
& \phi_p \cdot cons \cup \phi_p \cdot \phi_p \cdot \pi_2 \\
= & \{ \ \phi_p \cdot \phi_p = \phi_p \text{ (2.163) } \} \\
& \phi_p \cdot cons \cup \phi_p \cdot \pi_2 \\
= & \{ \text{ bilinearity of join (2.20) } \} \\
& \phi_p \cdot (cons \cup \pi_2) \\
\Box &
\end{array}
$$

Proof of (4.9):

$$
\begin{array}{ll}
& S \upharpoonright R \\
= & \{ \ S = [nil, \phi_p \cdot cons \cup \pi_2] \ \}
\end{array}
$$

$$
\begin{array}{ll}
& [nil, \phi_p \cdot cons \cup \pi_2] \upharpoonright R \\
= & \quad \{ \text{ coproduct shrinking (2.201); function shrinking (2.190) } \} \\
& [nil, \phi_p \cdot cons \cup \pi_2 \upharpoonright R] \\
= & \quad \{ \text{ claim: } (\phi_p \cdot cons \cup \pi_2) \upharpoonright R = p \cdot cons \rightarrow cons, \pi_2 \text{ (see below) } \} \\
& [nil, p \cdot cons \rightarrow cons, \pi_2] \\
\square &
\end{array}
$$

Claim:

$$
\begin{array}{ll}
& (\phi_p \cdot cons \cup \pi_2) \upharpoonright R \\
= & \quad \{ \text{ shrinking of join (2.200); division (2.97) } \} \\
& \phi_p \cdot cons \upharpoonright R \cap R \cdot \pi_2 \cup \pi_2 \upharpoonright R \cap R \,/\, (\phi_p \cdot cons)^\circ \\
= & \quad \{ \text{ function shrinking (2.190); since } \phi_p \cdot cons \text{ is simple, apply (2.189) } \} \\
& \phi_p \cdot cons \cap R \cdot \pi_2 \cup \pi_2 \cap R \,/\, (\phi_p \cdot cons)^\circ \\
= & \quad \{ \text{ claim: } \phi_p \cdot cons \cap R \cdot \pi_2 = \phi_p \cdot cons \ \} \\
& \phi_p \cdot cons \cup \pi_2 \cap R \,/\, (\phi_p \cdot cons)^\circ \\
= & \quad \{ \text{ claim: } \pi_2 \cap R \,/\, (\phi_p \cdot cons)^\circ = \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \ \} \\
& \phi_p \cdot cons \cup \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \\
= & \quad \{ \ \phi_p \cdot cons = cons \cdot \phi_{p \cdot cons} \text{ (2.164) } \} \\
& cons \cdot \phi_{p \cdot cons} \cup \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \\
= & \quad \{ \text{ biproduct absorption (2.85) } \} \\
& [cons, \pi_2] \cdot [\phi_{p \cdot cons}, \phi_{\neg\,(p \cdot cons)}]^\circ \\
= & \quad \{ \text{ definition of guard (2.169) } \} \\
& [cons, \pi_2] \cdot (p \cdot cons)? \\
= & \quad \{ \text{ McCarthy's conditional (2.168) } \} \\
& p \cdot cons \rightarrow cons, \pi_2 \\
\square &
\end{array}
$$

Subclaims:

$$
\begin{array}{ll}
& \phi_p \cdot cons \cap R \cdot \pi_2 = \phi_p \cdot cons \\
\equiv & \quad \{ \text{ circular equality (2.8); universal property of join (2.14) } \} \\
& \phi_p \cdot cons \subseteq \phi_p \cdot cons \cap R \cdot \pi_2 \\
\equiv & \quad \{ \text{ universal property of join (2.14) } \} \\
& \phi_p \cdot cons \subseteq R \cdot \pi_2
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \{ \text{ monotonicity (2.32) } \} \\
& cons \subseteq R \cdot \pi_2 \\
\equiv & \{ \text{ (4.19) } \} \\
& true \\
\Box &
\end{array}
$$

The next claim relies on the fact that wt is a positive-valued function, so, with $f = sum \cdot wt^*$, we have:

$$f \cdot cons \subseteq (>) \cdot f \cdot \pi_2 \tag{B.4}$$

$$\pi_2 \cap R \,/\, (\phi_p \cdot cons)^\circ = \pi_2 \cdot \phi_{\neg\,(p \cdot cons)}$$

Ping:

$$
\begin{array}{ll}
& \pi_2 \cap R \,/\, (\phi_p \cdot cons)^\circ \subseteq \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \\
\equiv & \{ \text{ converse; right division (2.103) } \} \\
& \pi_2 \cap (R \,/\, \phi_p) \cdot cons \subseteq \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \\
\equiv & \{ \text{ (B.1); meet distributes through join; left linearity and universal property of join (2.21, 2.14) } \} \\
& \left\{ \begin{array}{l} \pi_2 \cap \frac{p}{false} \cdot cons \subseteq \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \\ \pi_2 \cap R \cdot cons \subseteq \pi_2 \cdot \phi_{\neg\,(p \cdot cons)} \end{array} \right.
\end{array}
$$

First condition:

$$
\begin{array}{ll}
& \pi_2 \cap \dfrac{p}{false} \cdot cons \subseteq \pi_2 \cdot \phi_{\neg\, p \cdot cons} \\
\equiv & \{ \text{ definition of } \phi_{\neg\, p} \text{; distribute } \pi_2 \text{ (see side condition proof below) } \} \\
& \pi_2 \cap \dfrac{p}{false} \cdot cons \subseteq \pi_2 \cap \pi_2 \cdot \dfrac{p}{false} \cdot cons \\
\Leftarrow & \{ \text{ monotonicity (2.29, 2.27) } \} \\
& \dfrac{p}{false} \subseteq \pi_2 \cdot \dfrac{p}{false} \\
\Leftarrow & \{ \text{ symmetric division (2.112); monotonicity (2.27) } \} \\
& false^\circ \subseteq \pi_2 \cdot false^\circ \\
\equiv & \{ \text{ converse } \} \\
& false^\circ \subseteq (false \cdot \pi_2^\circ)^\circ \\
\equiv & \{ \textit{false} \text{ is constant and } \pi_2^\circ \text{ is entire, so apply (2.180) } \} \\
& false^\circ \subseteq false^\circ
\end{array}
$$

□

Side condition:

$$
\begin{array}{ll}
& \ker \pi_2 \cdot \dfrac{p \cdot cons}{false} \subseteq \dfrac{p \cdot cons}{false} \\
\Leftarrow & \quad \{ \text{ symmetric division (2.112); monotonicity (2.27) } \} \\
& \ker \pi_2 \cdot false^\circ \subseteq false^\circ \\
\equiv & \quad \{ \text{ shunting (2.58, 2.59) } \} \\
& false \cdot \ker \pi_2 \subseteq false \\
\equiv & \quad \{ \text{ } false \text{ is constant and } \pi_2 \text{ is a function, so } \ker \pi_2 \text{ is total, apply (2.180) } \} \\
& false \subseteq false \\
\end{array}
$$

□

Second condition:

$$
\begin{array}{ll}
& \pi_2 \cap R \cdot cons \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\equiv & \quad \{ \ R = (\geqslant)_f = f^\circ \cdot (\geqslant) \cdot f \text{ (2.6) } \} \\
& \pi_2 \cap f^\circ \cdot (\geqslant) \cdot f \cdot cons \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\Leftarrow & \quad \{ \text{ (B.4) } \} \\
& \pi_2 \cap f^\circ \cdot (\geqslant) \cdot (>) \cdot f \cdot \pi_2 \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\Leftarrow & \quad \{ \ (\geqslant) \cdot (>) \subseteq (>) \ \} \\
& \pi_2 \cap (>)_f \cdot \pi_2 \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\equiv & \quad \{ \ \pi_2 \text{ distributes through meet (2.25) because } \operatorname{img} \pi_2 \subseteq id \ \} \\
& (id \cap (>)_f) \cdot \pi_2 \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\Leftarrow & \quad \{ \text{ raise the lower side (2.32) with } id \subseteq (=)_f \ \} \\
& ((=)_f \cap (>)_f) \cdot \pi_2 \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\equiv & \quad \{ \text{ strictly greater and equal in value is impossible; } \bot \text{ is zero of composition (2.57) } \} \\
& \bot \subseteq \pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \\
\equiv & \quad \{ \ \bot \text{ below everything (2.51) } \} \\
& true \\
\end{array}
$$

□

Pong:

$$\pi_2 \cdot \phi_{\neg\ (p \cdot cons)} \subseteq \pi_2 \cap R \ / \ (\phi_p \cdot cons)^\circ$$

$$
\begin{array}{ll}
\equiv & \{ \text{ shunting (2.59) } \} \\
& \phi_{\neg\ (p \cdot cons)} \subseteq \pi_2^\circ \cdot (\pi_2 \cap R \ / \ (\phi_p \cdot cons)^\circ) \\
\equiv & \{ \text{ (2.24) because } \ker\ \pi_2^\circ \cdot \pi_2 \subseteq \pi_2 \ \} \\
& \phi_{\neg\ (p \cdot cons)} \subseteq \pi_2^\circ \cdot \pi_2 \cap \pi_2^\circ \cdot (R \ / \ (\phi_p \cdot cons)^\circ) \\
\Leftarrow & \{ \ \pi_2 \text{ is entire (2.42); definition of } \phi_{\neg\ p} \text{ (2.162) } \} \\
& id \cap \dfrac{false}{p \cdot cons} \subseteq id \cap \pi_2^\circ \cdot (R \ / \ (\phi_p \cdot cons)^\circ) \\
\equiv & \{ \text{ monotonicity (2.29) } \} \\
& \dfrac{false}{p \cdot cons} \subseteq \pi_2^\circ \cdot (R \ / \ (\phi_p \cdot cons)^\circ) \\
\equiv & \{ \text{ shunting (2.59); right division (2.108) } \} \\
& \pi_2 \cdot \dfrac{false}{p \cdot cons} \cdot (\phi_p \cdot cons)^\circ \subseteq R \\
\equiv & \{ \ \phi_p \cdot cons = cons \cdot \phi_{p \cdot cons} \text{ (2.164); converse } \} \\
& \pi_2 \cdot \dfrac{false}{p \cdot cons} \cdot \phi_{p \cdot cons} \cdot cons^\circ \subseteq R \\
\equiv & \{ \text{ definition of } \phi_p \text{ (2.159); monotonicity (2.32) } \} \\
& \pi_2 \cdot \dfrac{false}{p \cdot cons} \cdot \dfrac{p \cdot cons}{true} \cdot cons^\circ \subseteq R \\
\equiv & \{ \text{ shunting (2.58, 2.59); symmetric division (2.115) } \} \\
& \dfrac{false}{true} \subseteq \pi_2^\circ \cdot R \cdot cons \\
\equiv & \{ \ \frac{false}{true} = true^\circ \cdot false = \bot \text{ (4.11) } \} \\
& \bot \subseteq \pi_2^\circ \cdot R \cdot cons \\
\equiv & \{ \ \bot \text{ below everything (2.51) } \} \\
& true \\
\Box &
\end{array}
$$

IMPLEMENTING $SelectBy$ Basic definitions:

$$
\begin{aligned}
Select &= cons^\circ \cdot Perm \\
SelectBy\ R &= Select \upharpoonright R_{\pi_1} \\
fetch &= cons \cdot aux
\end{aligned}
$$

where

$$
\begin{aligned}
&aux\ (a,[\,]) = (a,[\,])\\
&aux\ (a,h:t)\\
&\quad \mid a \equiv h = (a,t)\\
&\quad \mid otherwise = \mathbf{let}\ (a',x) = aux\ (a,t)\ \mathbf{in}\ (a',h:x)
\end{aligned}
$$

Fetching an element from a list — meaning removing its first occurrence and adding it upfront — will either produce a permutation of the original list, or, in case the element is not found and thus not removed, the product of $cons$-ing it to the original list:

$$fetch \subseteq p \to Perm \cdot \pi_2, cons\ \mathbf{where}\ p\ (a,x) = a\ \epsilon\ x \tag{B.5}$$

Proving (B.5):

$$
\begin{aligned}
& fetch \subseteq p \to Perm \cdot \pi_2, cons\\
\equiv & \quad \{\ \text{shunting (2.59)}\ \}\\
& aux \subseteq cons^\circ \cdot (p \to Perm \cdot \pi_2, cons)\\
\equiv & \quad \{\ \text{McCarthy's conditional (2.170)}\ \}\\
& aux \subseteq cons^\circ \cdot (Perm \cdot \pi_2 \cdot \phi_p \cup cons \cdot \phi_{\neg\, p})\\
\equiv & \quad \{\ \text{join right linearity (2.20); } cons \text{ is injective (2.40), so } \ker\, cons = id\ \}\\
& aux \subseteq cons^\circ \cdot Perm \cdot \pi_2 \cdot \phi_p \cup \phi_{\neg\, p}\\
\equiv & \quad \{\ \text{let } aux = (\!|\, R\,|\!) \cdot (\!|\, S\,|\!)^\circ \text{ — see below}\ \}\\
& (\!|\, R\,|\!) \cdot (\!|\, S\,|\!)^\circ \subseteq \underbrace{cons^\circ \cdot Perm \cdot \pi_2 \cdot \phi_p \cup \phi_{\neg\, p}}_{X}\\
\Leftarrow & \quad \{\ \text{least fixpoint rule (2.220)}\ \}\\
& R \cdot \mathbb{F}\ X \cdot S^\circ \subseteq X
\end{aligned}
$$

Note that aux is a $\mathbb{F}$-hylomorphism, for $\mathbb{F}\ f = id + id \times f$, whose divide step $S^\circ = g$ where

$$
\begin{aligned}
&g\ (a,[\,]) = i_1\ (a,[\,])\\
&g\ (a,h:t)\\
&\quad \mid a \equiv h = i_1\ (a,t)\\
&\quad \mid otherwise = i_2\ (h,(a,t))
\end{aligned}
$$

Its conquer step R is function $f = [id, k]\ \mathbf{where}\ k\ (h,(a,x)) = (a,h:x)$. Clearly, $S \subseteq [s_1, k \cdot \phi_r]$ where:

$$
\begin{aligned}
&s_1 = q \to id, \langle \pi_1, cons \rangle\\
&q\ (a,x) = x \equiv [\,]\\
&r\ (h,(a,x)) = h \not\equiv a
\end{aligned}
$$

Then:

$$
\begin{array}{ll}
& R \cdot \mathbb{F}\ X \cdot S^\circ \subseteq X \\
\Leftarrow & \quad \{\ S \subseteq [s_1, k \cdot \phi_r]\text{; definitions of } R \text{ and } \mathbb{F}\ \} \\
& [id, k] \cdot (id + id \times X) \cdot [s_1, k \cdot \phi_r]^\circ \subseteq X \\
\equiv & \quad \{\ \text{direct sum and biproduct absorption (2.87, 2.85); universal property of join (2.14)}\ \} \\
& \left\{ \begin{array}{l} s_1^\circ \subseteq X \\ k \cdot (id \times X) \cdot \phi_r \cdot k^\circ \subseteq X \end{array} \right.
\end{array}
$$

Handling $s_1^\circ \subseteq X$:

$$
\begin{array}{ll}
& s_1^\circ \subseteq X \\
\equiv & \quad \{\ \text{McCarthy's conditional (2.170); converse}\ \} \\
& \phi_q \cup \phi_{\neg\, q} \cdot \langle \pi_1, cons \rangle^\circ \subseteq X \\
\Leftarrow & \quad \{\ \text{universal property of join (2.14); shunting (2.58); raise lower side (2.32) with (2.160)}\ \} \\
& \left\{ \begin{array}{l} \phi_q \subseteq \phi_{\neg\, p} \\ id \subseteq X \cdot \langle \pi_1, cons \rangle \end{array} \right. \\
\Leftarrow & \quad \{\ \text{go pointwise in first equation and simplify; expand } X\ \} \\
& \left\{ \begin{array}{l} x = [\,] \Rightarrow \neg\ (a \in x) \\ id \subseteq cons^\circ \cdot Perm \cdot \pi_2 \cdot \langle \pi_1, cons \rangle \end{array} \right. \\
\equiv & \quad \{\ \text{empty list contains no elements; shunting (2.59); pairing cancellation (2.65)}\ \} \\
& cons \subseteq Perm \cdot cons \\
\Leftarrow & \quad \{\ Perm \text{ is reflexive (2.35), so lower upper side (2.31)}\ \} \\
& cons \subseteq cons \\
& \square
\end{array}
$$

Handling $k \cdot (id \times X) \cdot \phi_r \cdot k^\circ \subseteq X$:

$$
\begin{array}{ll}
& k \cdot (id \times X) \cdot \phi_r \cdot k^\circ \subseteq X \\
\equiv & \quad \{\ \text{shunting (2.58, 2.59)}\ \} \\
& (id \times X) \cdot \phi_r \subseteq k^\circ \cdot X \cdot k \\
\Leftarrow & \quad \{\ \text{bifunctors; properties of join (2.21, 2.14); raise lower side (2.32) with (2.160)}\ \} \\
& \left\{ \begin{array}{l} id \times (cons^\circ \cdot Perm \cdot \pi_2 \cdot \phi_p) \subseteq k^\circ \cdot X \cdot k \\ (id \times \phi_{\neg\, p}) \cdot \phi_r \subseteq k^\circ \cdot X \cdot k \end{array} \right. \\
\Leftarrow & \quad \{\ \text{lower upper side (2.31); introduce points}\ \}
\end{array}
$$

$$
\begin{cases} (a' : x')\ (Perm \cdot \pi_2 \cdot \phi_p)\ (a, x) \Rightarrow k\ (h, (a', x'))\ X\ (k\ (h, (a, x))) \\ (id \times \phi_{\neg\, p}) \cdot \phi_r \subseteq k^\circ \cdot \phi_{\neg\, p} \cdot k \end{cases}
$$

$\equiv$ { $k\ (h, (a, x)) = (a, h : x)$; use p and r in ϕ_p and ϕ_r directly }

$$
\begin{cases} a \in x \wedge (a' : x')\ Perm\ x \Rightarrow (a', h : x')\ X\ (a, h : x) \\ \neg\ (a \in x) \wedge a \not\equiv h \Rightarrow (a, (h : x))\ \phi_{\neg\, p}\ (a, (h : x)) \end{cases}
$$

$\equiv$ { expand X }

$$
\begin{cases} a \in x \wedge (a' : x')\ Perm\ x \Rightarrow a \in (h : x) \wedge (a' : h : x')\ Perm\ (h : x) \\ a \in (h : x) \Rightarrow a = h \vee a \in x \end{cases}
$$

$\equiv$ { definition of ϵ; expand $Perm$; $a \in x \Rightarrow a \in (h : x)$ }

$$
\hat{a'} + bag\ x' = bag\ x \Rightarrow \hat{a'} + \hat{h'} + bag\ x' = \hat{h'} + bag\ x
$$

$\equiv$ { substitute $bag\ x$; $(+)$ is commutative }

$$
\hat{a'} + \hat{h} + bag\ x' = \hat{a'} + \hat{h} + bag\ x'
$$

□

With this we state that removing an element that is a member of the list and then adding it at the head will produce a permutation of the original list:

$$
fetch \cdot \langle \epsilon, id \rangle \subseteq Perm \tag{B.6}
$$

Proof:

$$
fetch \cdot \langle \epsilon, id \rangle
$$

$\subseteq$ { (B.5) }

$$
(Perm \cdot \pi_2 \cdot \phi_p \cup \phi_{\neg\, p}) \cdot \langle \epsilon, id \rangle
$$

$\subseteq$ { left linearity of join (2.21); pairing cancellation (2.63) }

$$
Perm \cup \phi_{\neg\, p} \cdot \langle \epsilon, id \rangle
$$

$=$ { claim: $\phi_{\neg\, p} \cdot \langle \epsilon, id \rangle = \bot$, see below }

$$
Perm
$$

□

The claim above is immediate: $\phi_{\neg\, p} \cdot \langle \epsilon, id \rangle = \bot \Leftrightarrow \neg \langle \exists\, a : \neg\, (a \in x) :\ a \in x \rangle$

Select is non-deterministic in that it allows any permutation of the remaining list. We are able to refine this to a function that chooses just one of them, specifically, the list that arises from removing the first occurrence of the selected element.

$$
aux \cdot \langle \epsilon, id \rangle \subseteq Select \tag{B.7}
$$

Proof:

$$
\begin{array}{ll}
& aux \cdot \langle \epsilon, id \rangle \subseteq Select \\
\equiv & \quad \{ \text{ definition of } Select \text{ (3.19) } \} \\
& aux \cdot \langle \epsilon, id \rangle \subseteq cons^\circ \cdot Perm \\
\equiv & \quad \{ \text{ shunting (2.59) } \} \\
& cons \cdot aux \cdot \langle \epsilon, id \rangle \subseteq Perm \\
\equiv & \quad \{ \text{ (B.6) } \} \\
& true \\
\Box &
\end{array}
$$

Using this, we calculate $SelectBy\ R$:

$$
\begin{array}{ll}
& SelectBy\ R \\
= & \quad \{ \text{ definition of } SelectBy \ \} \\
& Select \upharpoonright R_{\pi_1} \\
\supseteq & \quad \{ \text{ shrinking monotonicity (2.195) } \} \\
& aux \cdot \langle \epsilon, id \rangle \upharpoonright R_{\pi_1} \\
= & \quad \{ \text{ function shrinking (2.192); pairing cancellation (2.64) } \} \\
& aux \cdot (\langle \epsilon, id \rangle \upharpoonright R_{\pi_1}) \\
= & \quad \{ \ R_{\pi_1} = R \times \top \text{ in pairing shrinking (2.203) } \} \\
& aux \cdot \langle \epsilon \upharpoonright R, id \rangle
\end{array}
$$

If R is a connected preorder, we refine even further by using the $minlist$ function, yielding an executable function:

$$SelectBy\ R \ \supseteq \ selectMin\ R = aux \cdot \langle minlist\ R, id \rangle \ \Leftarrow \ R \text{ is connected}$$

(Recall (3.28).)

THE COLOR PROBLEM Sublists of the empty list are no worse with respect to R than empty list:

$$nil \cdot \phi_{null} \cdot H^\circ \subseteq R$$

Proof:

$$
\begin{array}{ll}
& nil \cdot \phi_{null} \cdot H^\circ \subseteq R \\
\equiv & \quad \{ \ \phi_{null} = nil \cdot nil^\circ \text{ (3.9) } \} \\
& nil \cdot nil \cdot nil^\circ \cdot H^\circ \subseteq R
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{ \text{ converse; } H = \phi_p \cdot (\sqsubseteq) \cdot Perm \ \} \\
& nil \cdot nil \cdot (\phi_p \cdot (\sqsubseteq) \cdot Perm \cdot nil)^\circ \subseteq R \\
\equiv & \{ \ (3.15); (3.35); \text{monotonicity } (2.32) \ \} \\
& nil \cdot nil \cdot nil^\circ \subseteq R \\
\equiv & \{ \ nil \text{ is entire and a constant function } (2.180) \ \} \\
& nil \cdot nil^\circ \subseteq R \\
\Leftarrow & \{ \ nil \text{ is simple } (2.41) \ \} \\
& id \subseteq R \\
\equiv & \{ \ R \text{ is reflexive } (2.35) \ \} \\
& true \\
\Box &
\end{array}
$$

Proof of (4.21):

$$
\begin{array}{ll}
& \phi_p \cdot (\trianglelefteq) \cdot cons = \phi_p \cdot (\trianglelefteq) \cdot cons \cdot (id \times \phi_p \cdot (\trianglelefteq)) \\
\equiv & \{ \text{ definition of } (\trianglelefteq) \ (3.43, 3.47) \ \} \\
& \phi_p \cdot (\sqsubseteq) \cdot Perm \cdot cons = \phi_p \cdot (\sqsubseteq) \cdot Perm \cdot cons \cdot (id \times \phi_p \cdot Perm \cdot (\sqsubseteq)) \\
\equiv & \{ \ p \text{ is stable, so } (3.34); \text{bifunctors; } (3.14) \ \} \\
& \phi_p \cdot (\sqsubseteq) \cdot Perm \cdot cons = \phi_p \cdot (\sqsubseteq) \cdot Perm \cdot cons \cdot (id \times \phi_p \cdot (\sqsubseteq)) \\
\Leftarrow & \{ \ (3.47); (3.34); \text{monotonicity } (2.27) \ \} \\
& \phi_p \cdot (\sqsubseteq) \cdot cons = \phi_p \cdot (\sqsubseteq) \cdot cons \cdot (id \times \phi_p \cdot (\sqsubseteq)) \\
\equiv & \{ \ \phi_p \cdot (\sqsubseteq) \text{ is a catamorphism } (4.8) \ \} \\
& (\phi_p \cdot cons \cup \pi_2) \cdot (id \times \phi_p \cdot (\sqsubseteq)) = (\phi_p \cdot cons \cup \pi_2) \cdot (id \times \phi_p \cdot (\sqsubseteq)) \cdot (id \times \phi_p \cdot (\sqsubseteq)) \\
\Leftarrow & \{ \ \text{monotonicity } (2.27); \times \text{ is a bifunctor } \} \\
& (id \times \phi_p \cdot (\sqsubseteq)) = (id \times \phi_p \cdot (\sqsubseteq) \cdot \phi_p \cdot (\sqsubseteq)) \\
\equiv & \{ \ \text{monotonicity; circular equality } (2.8) \ \} \\
& \left\{ \begin{array}{l} \phi_p \cdot (\sqsubseteq) \subseteq \phi_p \cdot (\sqsubseteq) \cdot \phi_p \cdot (\sqsubseteq) \\ \phi_p \cdot (\sqsubseteq) \cdot \phi_p \cdot (\sqsubseteq) \subseteq \phi_p \cdot (\sqsubseteq) \end{array} \right.
\end{array}
$$

First condition:

$$
\begin{array}{ll}
& \phi_p \cdot (\sqsubseteq) \subseteq \phi_p \cdot (\sqsubseteq) \cdot \phi_p \cdot (\sqsubseteq) \\
\Leftarrow & \{ \ (\sqsubseteq) \text{ is reflexive } (2.35) \ \} \\
& \phi_p \cdot (\sqsubseteq) \subseteq \phi_p \cdot \phi_p \cdot (\sqsubseteq) \\
\equiv & \{ \ \phi_p \cdot \phi_p = \phi_p \ (2.163) \ \}
\end{array}
$$

$$\phi_p \cdot (\sqsubseteq) \subseteq \phi_p \cdot (\sqsubseteq)$$

□

Second condition:

$$\begin{array}{ll}
& \phi_p \cdot (\sqsubseteq) \cdot \phi_p \cdot (\sqsubseteq) \subseteq \phi_p \cdot (\sqsubseteq) \\
\Leftarrow & \quad \{ \text{ monotonicity (2.27); raise the lower side (2.32) } \} \\
& (\sqsubseteq) \cdot (\sqsubseteq) \subseteq (\sqsubseteq) \\
\equiv & \quad \{ \text{ } (\sqsubseteq) \text{ is transitive (2.37) } \} \\
& true
\end{array}$$

□

Proof of (4.22):

$$\begin{array}{ll}
& sum \cdot v^* \cdot \phi_p \cdot (\trianglelefteq) \subseteq (\leqslant) \cdot sum \cdot v^* \\
\equiv & \quad \{ \text{ monotonicity (2.32); definition of } (\trianglelefteq) \text{ (3.47) } \} \\
& sum \cdot v^* \cdot Perm \cdot (\sqsubseteq) \subseteq (\leqslant) \cdot sum \cdot v^* \\
\equiv & \quad \{ \text{ } v^* \cdot Perm = Perm \cdot v^* \text{ (3.18); } sum \text{ is stable (3.32) } \} \\
& sum \cdot v^* \cdot (\sqsubseteq) \subseteq (\leqslant) \cdot sum \cdot v^* \\
\equiv & \quad \{ \text{ both } sum \cdot v^* \text{ and } sum \cdot v^* \cdot (\sqsubseteq) \text{ are catamorphisms (see proof below) } \} \\
& (\!| [\underline{0}, (add \cup \pi_2) \cdot (v \times id)] |\!) \subseteq (\leqslant) \cdot (\!| [\underline{0}, add \cdot (v \times id)] |\!) \\
\Leftarrow & \quad \{ \text{ catamorphism fusion (2.217) } \} \\
& [\underline{0}, (add \cup \pi_2) \cdot (v \times id)] \cdot \mathbb{F} (\leqslant) \subseteq (\leqslant) \cdot [\underline{0}, add \cdot (v \times id)] \\
\equiv & \quad \{ \text{ coproducts } \} \\
& \left\{ \begin{array}{l} \underline{0} \subseteq (\leqslant) \cdot \underline{0} \\ (add \cup \pi_2) \cdot (v \times id) \cdot (id \times (\leqslant)) \subseteq (\leqslant) \cdot add \cdot (v \times id) \end{array} \right. \\
\Leftarrow & \quad \{ \text{ biproducts; monotonicity (2.27) } \} \\
& \left\{ \begin{array}{l} id \subseteq (\leqslant) \\ (add \cup \pi_2) \cdot (id \times (\leqslant)) \cdot (v \times id) \subseteq (\leqslant) \cdot add \cdot (v \times id) \end{array} \right. \\
\equiv & \quad \{ \text{ } (\leqslant) \text{ is reflexive (2.35) (twice), so after (2.32), and join properties (2.21) and (2.14), we get } \} \\
& \left\{ \begin{array}{l} add \cdot ((\leqslant) \times (\leqslant)) \subseteq (\leqslant) \cdot add \\ \pi_2 \cdot ((\leqslant) \times (\leqslant)) \cdot (v \times id) \subseteq (\leqslant) \cdot add \cdot (v \times id) \end{array} \right. \\
\Leftarrow & \quad \{ \text{ } add \text{ is monotonic with respect to } (\leqslant); \text{ free theorem of } \pi_2 \text{ (twice) } \} \\
& (\leqslant) \cdot \pi_2 \subseteq (\leqslant) \cdot add \cdot (v \times id) \\
\equiv & \quad \{ \text{ convert to pointwise } \}
\end{array}$$

$$
\begin{array}{ll}
& a \leqslant c \Rightarrow a \leqslant v\ b + c \\
\equiv & \quad \{\ \text{since } v\ b \text{ is always positive}\ \} \\
& true \\
\square &
\end{array}
$$

Proof that $sum \cdot v^*$ is a catamorphism:

$$
\begin{array}{ll}
& sum \cdot v^* = (\!|\, [\underline{0}, add \cdot (v \times id)]\, |\!) \\
\equiv & \quad \{\ \text{catamorphism fusion (2.213)}\ \} \\
& sum \cdot \mathsf{in} \cdot \mathbb{B}\ (v, id) = [\underline{0}, add \cdot (v \times id)] \cdot \mathbb{F}\ sum \\
\equiv & \quad \{\ \text{catamorphism (2.204)}\ \} \\
& [\underline{0}, add] \cdot \mathbb{F}\ sum \cdot \mathbb{B}\ (v, id) = [\underline{0}, add \cdot (v \times id)] \cdot \mathbb{F}\ sum \\
\equiv & \quad \{\ \mathbb{F}\ X = \mathbb{B}\ (id, X)\text{; bifunctors; direct sum absorption (2.87)}\ \} \\
& [\underline{0}, add \cdot (v \times sum)] = [\underline{0}, add \cdot (v \times sum)] \\
\square &
\end{array}
$$

Now use that result to prove $sum \cdot v^* \cdot (\sqsubseteq)$ is also a catamorphism:

$$
\begin{array}{ll}
& sum \cdot v^* \cdot (\sqsubseteq) = (\!|\, [\underline{0}, (add \cup \pi_2) \cdot (v \times id)]\, |\!) \\
\equiv & \quad \{\ \text{catamorphism fusion (2.213)}\ \} \\
& sum \cdot v^* \cdot [nil, cons \cup \pi_2] = [\underline{0}, (add \cup \pi_2) \cdot (v \times id)] \cdot \mathbb{F}\ (sum \cdot v^*) \\
\equiv & \quad \{\ \text{coproduct fusion (2.83); direct sum absorption (2.87); coproduct equality (2.80)}\ \} \\
& \left\{ \begin{array}{l} sum \cdot v^* \cdot nil = \underline{0} \\ sum \cdot v^* \cdot (cons \cup \pi_2) = (add \cup \pi_2) \cdot (v \times id) \cdot (id \times sum \cdot v^*) \end{array} \right. \\
\equiv & \quad \{\ sum \cdot v^* \cdot nil = sum \cdot nil = \underline{0}\ \} \\
& sum \cdot v^* \cdot (cons \cup \pi_2) = (add \cup \pi_2) \cdot (v \times id) \cdot (id \times sum \cdot v^*)
\end{array}
$$

Proceeding by direct equality (justifications involving bilinearity of join (2.21, 2.20) are omitted):

$$
\begin{array}{ll}
& sum \cdot v^* \cdot (cons \cup \pi_2) \\
= & \quad \{\ \text{free theorem of } cons \text{ (3.5), expand } v^* \cdot cons \text{ with (3.4); Kronecker product cancellation (2.75)}\ \} \\
& sum \cdot (cons \cup \pi_2) \cdot (v \times v^*) \\
= & \quad \{\ sum \cdot cons = add \cdot (id \times sum)\text{; Kronecker product cancellation (2.75)}\ \} \\
& (add \cup \pi_2) \cdot (id \times sum) \cdot (v \times v^*) \\
= & \quad \{\ \text{biproducts}\ \} \\
& (add \cup \pi_2) \cdot (v \times id) \cdot (id \times sum \cdot v^*)
\end{array}
$$

□

INVARIANT OF THE COLOR PROBLEM The derived solution satisfies the original specification including the invariant:

$$C \subseteq (\phi_p \cdot (\trianglelefteq) \cap I) \upharpoonright R$$

where $C = (\!|\, [nil, aug] \,|\!) \cdot (\!|\, [nil, (SelectBy\ R)^\circ] \,|\!)^\circ$. Proof:

$$\begin{array}{ll} & C \subseteq (\phi_p \cdot (\trianglelefteq) \cap I) \upharpoonright R \\ \Leftarrow & \quad \{ \text{ universal property (2.185); relational meet (2.14); monotonicity (2.32) } \} \\ & \left\{ \begin{array}{l} C \subseteq \phi_p \cdot (\trianglelefteq) \\ C \subseteq I \\ C \cdot (\phi_p \cdot (\trianglelefteq))^\circ \subseteq R \end{array} \right. \end{array}$$

Through the main derivation of the color problem, we have proven $C \subseteq \phi_p \cdot (\trianglelefteq) \upharpoonright R$, so the first and third conditions are easily solved applying the shrinking cancellation laws (2.187) and (2.188).

$$\begin{array}{ll} & C \subseteq I \\ \Leftarrow & \quad \{ \text{ definition of } C\text{; strengthen invariant by lowering the upper side (2.31) } \} \\ & (\!|\, [nil, aug] \,|\!) \cdot (\!|\, [nil, (SelectBy\ R)^\circ] \,|\!)^\circ \subseteq \phi_p \cdot I \\ \Leftarrow & \quad \{ \text{ hylomorphism least fixpoint (2.220); } SelectBy\ R \subseteq Select \text{ by (2.187) } \} \\ & [nil, aug] \cdot \mathbb{F}\ (\phi_p \cdot I) \cdot [nil, Select^\circ]^\circ \subseteq \phi_p \cdot I \end{array}$$

Unfolding the left side using the properties of relational biproducts (2.87, 2.85) yields two conditions:

$$nil \cdot nil^\circ \subseteq \phi_p \cdot I \tag{B.8}$$

$$aug \cdot (id \times (\phi_p \cdot I)) \cdot Select \subseteq \phi_p \cdot I \tag{B.9}$$

Proof of (B.8):

$$\begin{array}{ll} & nil \cdot nil^\circ \subseteq \phi_p \cdot I \\ \equiv & \quad \{ \text{ shunting (2.58); converse; } \phi_p \cdot nil = nil \text{ (4.2) } \} \\ & nil^\circ \subseteq nil^\circ \cdot I \\ \equiv & \quad \{ \ I = \frac{c \cdot \epsilon}{c \cdot \epsilon} = \frac{\Lambda(c \cdot \epsilon)}{\Lambda(c \cdot \epsilon)}\text{; converses } \} \\ & nil \subseteq \dfrac{\Lambda(c \cdot \epsilon)}{\Lambda(c \cdot \epsilon)} \cdot nil \\ \equiv & \quad \{ \text{ symmetric division (2.112); shunting (2.58) } \} \\ & \Lambda(c \cdot \epsilon) \cdot nil \subseteq \Lambda(c \cdot \epsilon) \cdot nil \end{array}$$

□

For (B.9), it is more convenient to separate the proof in two parts using (2.167):

$$
\begin{array}{ll}
& aug \cdot (id \times (\phi_p \cdot I)) \cdot Select \subseteq \phi_p \cdot I \\
\equiv & \quad \{\ I, \text{ a equivalence relation, is entire and surjective, so apply (2.167)}\ \} \\
& aug \cdot (id \times I \cap \dfrac{true}{p}) \cdot Select \subseteq I \cap \dfrac{true}{p}
\end{array}
$$

Applying the universal property of relational meet (2.13) yields two more conditions:

$$aug \cdot (id \times I \cap \frac{true}{p}) \cdot Select \subseteq \frac{true}{p} \tag{B.10}$$

$$aug \cdot (id \times I \cap \frac{true}{p}) \cdot Select \subseteq I \tag{B.11}$$

Proof of (B.10):

$$
\begin{array}{ll}
& aug \cdot (id \times I \cap \dfrac{true}{p}) \cdot Select \subseteq \dfrac{true}{p} \\
\Leftarrow & \quad \{\ \text{monotonicity (2.32)}\ \} \\
& aug \cdot (id \times \dfrac{true}{p}) \cdot Select \subseteq \dfrac{true}{p} \\
\equiv & \quad \{\ \text{McCarthy conditional (2.170); relational join (2.21, 2.14)}\ \} \\
& \left\{ \begin{array}{l} \pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times \frac{true}{p}) \cdot Select \subseteq \frac{true}{p} \\ cons \cdot \phi_{p \cdot cons} \cdot (id \times \frac{true}{p}) \cdot Select \subseteq \frac{true}{p} \end{array} \right. \\
\Leftarrow & \quad \{\ cons \cdot \phi_{p \cdot cons} = \phi_p \cdot cons \text{ (2.164); monotonicity (2.32)}\ \} \\
& \left\{ \begin{array}{l} \pi_2 \cdot (id \times \frac{true}{p}) \cdot Select \subseteq \frac{true}{p} \\ \phi_p \cdot cons \cdot (id \times \frac{true}{p}) \cdot Select \subseteq \frac{true}{p} \end{array} \right. \\
\Leftarrow & \quad \{\ \text{Kronecker product cancellation (2.75); } \phi_p \subseteq \frac{true}{p} \text{ (2.161)}\ \} \\
& \left\{ \begin{array}{l} \frac{true}{p} \cdot \pi_2 \cdot Select \subseteq \frac{true}{p} \\ \frac{true}{p} \cdot cons \cdot (id \times \frac{true}{p}) \cdot Select \subseteq \frac{true}{p} \end{array} \right. \\
\Leftarrow & \quad \{\ \text{symmetric division (2.112; monotonicity (2.27) (twice)}\ \} \\
& \left\{ \begin{array}{l} true \cdot \pi_2 \cdot Select \subseteq true \\ true \cdot cons \cdot (id \times \frac{true}{p}) \cdot Select \subseteq true \end{array} \right. \\
\equiv & \quad \{\ \text{shunting (2.59) and symmetric division (2.112) (twice); } \frac{true}{true} = \top \text{ (2.182)}\ \} \\
& \left\{ \begin{array}{l} \pi_2 \cdot Select \subseteq \top \\ cons \cdot (id \times \frac{true}{p}) \cdot Select \subseteq \top \end{array} \right.
\end{array}
$$

$$\equiv \quad \{ \ \top \text{ above everything (2.51) (twice)} \ \}$$
$$true$$
$$\square$$

Proof of (B.11):

$$aug \cdot (id \times I \cap \frac{true}{p}) \cdot Select \subseteq I$$
$$\equiv \quad \{ \ \text{McCarthy conditional (2.170); relational join (2.21, 2.14)} \ \}$$
$$\begin{cases} \pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times I \cap \frac{true}{p}) \cdot Select \subseteq I \\ cons \cdot \phi_{p \cdot cons} \cdot (id \times I \cap \frac{true}{p}) \cdot Select \subseteq I \end{cases}$$

Expanding the definition of I, it is possible to apply the universal property of meet (2.13). For each of the resulting 4 conditions, monotonicity (2.32) is applied to obtain the useful part of the invariant on the lower side.

$$\pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \setminus c \cdot \epsilon \tag{B.12}$$
$$\pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ \cap \frac{true}{p}) \cdot Select \subseteq (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ \tag{B.13}$$
$$cons \cdot \phi_{p \cdot cons} \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \setminus c \cdot \epsilon \tag{B.14}$$
$$cons \cdot \phi_{p \cdot cons} \cdot (id \times (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ) \cdot Select \subseteq (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ \tag{B.15}$$

Proof of (B.12):

$$\pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \setminus c \cdot \epsilon$$
$$\Leftarrow \quad \{ \ \text{monotonicity (2.32); Kronecker product cancellation (2.75)} \ \}$$
$$(c \cdot \epsilon \setminus c \cdot \epsilon) \cdot \pi_2 \cdot Select \subseteq c \cdot \epsilon \setminus c \cdot \epsilon$$
$$\Leftarrow \quad \{ \ \text{left division (2.91, 2.93)} \ \}$$
$$c \cdot \epsilon \cdot \pi_2 \cdot Select \subseteq c \cdot \epsilon$$
$$\Leftarrow \quad \{ \ (3.26) \ \}$$
$$c \cdot \epsilon \subseteq c \cdot \epsilon$$
$$\square$$

Proof of (B.13):

$$\pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ \cap \frac{true}{p}) \cdot Select \subseteq (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ$$
$$\Leftarrow \quad \{ \ \text{right division (2.108)} \ \}$$
$$\pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times (c \cdot \epsilon)^\circ \,/\, (c \cdot \epsilon)^\circ \cap \frac{true}{p}) \cdot Select \cdot (c \cdot \epsilon)^\circ \subseteq (c \cdot \epsilon)^\circ$$

$$
\begin{array}{ll}
\equiv & \{\ \text{(B.17) (for any relation } R,\ R \mathbin{/} R \text{ is reflexive)}\ \} \\
& \pi_2 \cdot \phi_{\neg\, p \cdot cons} \cdot (id \times \phi_p) \cdot (id \times (c \cdot \epsilon)^\circ \mathbin{/} (c \cdot \epsilon)^\circ) \cdot Select \cdot (c \cdot \epsilon)^\circ \subseteq (c \cdot \epsilon)^\circ \\
\equiv & \{\ \text{(B.16); shunting (2.59); converses}\ \} \\
& (c \cdot \pi_1 \cup c \cdot \epsilon \cdot \pi_2) \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot (id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq c \cdot \epsilon \cdot \pi_2 \\
\equiv & \{\ \text{left linearity of join (2.21); universal property of join (2.14)}\ \} \\
& \left\{ \begin{array}{l} c \cdot \pi_1 \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot (id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq c \cdot \epsilon \cdot \pi_2 \\ c \cdot \epsilon \cdot \pi_2 \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot (id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq c \cdot \epsilon \cdot \pi_2 \end{array} \right. \\
\Leftarrow & \{\ \text{free theorems of } \pi_1 \text{ and } \pi_2 \text{ (2.72, 2.73); left division cancellation (2.93)}\ \} \\
& \left\{ \begin{array}{l} c \cdot \pi_1 \cdot (id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq c \cdot \epsilon \cdot \pi_2 \\ c \cdot \epsilon \cdot \pi_2 \cdot (id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq c \cdot \epsilon \cdot \pi_2 \end{array} \right. \\
\Leftarrow & \{\ \text{monotonicity (2.32) with } \phi_p \subseteq id\ \} \\
& c \cdot \pi_1 \cdot (id \times \phi_p) \cdot \phi_{\neg\, p \cdot cons} \subseteq c \cdot \epsilon \cdot \pi_2 \\
\Leftarrow & \{\ \text{(4.18)}\ \} \\
& c \cdot \pi_1 \cdot (c \cdot \pi_1)^\circ \cdot c \cdot \epsilon \cdot \pi_2 \subseteq c \cdot \epsilon \cdot \pi_2 \\
\Leftarrow & \{\ c \cdot \pi_1 \text{ is simple, so } \mathrm{img}\ c \cdot \pi_1 \subseteq id \text{ (2.41)}\ \} \\
& c \cdot \epsilon \cdot \pi_2 \subseteq c \cdot \epsilon \cdot \pi_2 \\
\Box &
\end{array}
$$

Proof of (B.14):

$$
\begin{array}{ll}
& cons \cdot \phi_{p \cdot cons} \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \setminus c \cdot \epsilon \\
\equiv & \{\ \text{left division (2.91); (2.164); monotonicity (2.32)}\ \} \\
& c \cdot \epsilon \cdot cons \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \\
\equiv & \{\ \text{(3.3); bilinearity of join (2.21, 2.20); universal property of join (2.14)}\ \} \\
& \left\{ \begin{array}{l} c \cdot \pi_1 \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \\ c \cdot \epsilon \cdot \pi_2 \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \cdot Select \subseteq c \cdot \epsilon \end{array} \right. \\
\Leftarrow & \{\ \text{free theorems of } \pi_1 \text{ and } \pi_2 \text{ (2.72, 2.73); left division cancellation (2.93)}\ \} \\
& \left\{ \begin{array}{l} c \cdot \pi_1 \cdot Select \subseteq c \cdot \epsilon \\ c \cdot \epsilon \cdot \pi_2 \cdot Select \subseteq c \cdot \epsilon \end{array} \right. \\
\Leftarrow & \{\ \text{(3.25, 3.26)}\ \} \\
& \left\{ \begin{array}{l} c \cdot \epsilon \subseteq c \cdot \epsilon \\ c \cdot \epsilon \subseteq c \cdot \epsilon \end{array} \right. \\
\Box &
\end{array}
$$

Proof of (B.15):

$$
\begin{array}{ll}
& cons \cdot \phi_{p \cdot cons} \cdot (id \times (c \cdot \epsilon)^\circ \mathbin{/} (c \cdot \epsilon)^\circ) \cdot Select \subseteq (c \cdot \epsilon)^\circ \mathbin{/} (c \cdot \epsilon)^\circ \\
\Leftarrow & \quad \{ \text{ monotonicity (2.32); right division (2.108) } \} \\
& cons \cdot (id \times (c \cdot \epsilon)^\circ \mathbin{/} (c \cdot \epsilon)^\circ) \cdot Select \cdot (c \cdot \epsilon)^\circ \subseteq (c \cdot \epsilon)^\circ \\
\equiv & \quad \{ \text{ (B.16); shunting (2.59); converses } \} \\
& (c \cdot \pi_1 \cup c \cdot \epsilon \cdot \pi_2) \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \subseteq c \cdot \epsilon \cdot cons \\
\equiv & \quad \{ \text{ left linearity of join (2.21); universal property of join (2.14) } \} \\
& \left\{ \begin{array}{l} c \cdot \pi_1 \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \subseteq c \cdot \epsilon \cdot cons \\ c \cdot \epsilon \cdot \pi_2 \cdot (id \times c \cdot \epsilon \setminus c \cdot \epsilon) \subseteq c \cdot \epsilon \cdot cons \end{array} \right. \\
\Leftarrow & \quad \{ \text{ free theorems of } \pi_1 \text{ and } \pi_2 \text{ (2.72, 2.73); left division cancellation (2.93) } \} \\
& \left\{ \begin{array}{l} c \cdot \pi_1 \subseteq c \cdot \epsilon \cdot cons \\ c \cdot \epsilon \cdot \pi_2 \subseteq c \cdot \epsilon \cdot cons \end{array} \right. \\
\equiv & \quad \{ \text{ monotonicity (2.27) (twice); universal property of join (2.14) } \} \\
& \pi_1 \cup \epsilon \cdot \pi_2 \subseteq \epsilon \cdot cons \\
\equiv & \quad \{ \text{ (3.3) } \} \\
& true \\
\Box &
\end{array}
$$

Claims:

$$Select \cdot (c \cdot \epsilon)^\circ = (c \cdot \pi_1)^\circ \cup (c \cdot \epsilon \cdot \pi_2)^\circ \tag{B.16}$$

$$R \cap \frac{true}{p} = \phi_p \cdot R \quad \Leftarrow \quad R \text{ is reflexive} \tag{B.17}$$

Proof of (B.16):

$$
\begin{array}{ll}
& Select \cdot (c \cdot \epsilon)^\circ \\
= & \quad \{ \text{ definition of } Select \text{ (3.19); } Perm = Perm^\circ \text{ (3.13); converse } \} \\
& (c \cdot \epsilon \cdot Perm \cdot cons)^\circ \\
= & \quad \{ \text{ } \epsilon \cdot Perm = \epsilon \text{ (3.31); (3.3) } \} \\
& (c \cdot (\pi_1 \cup \epsilon \cdot \pi_2))^\circ \\
= & \quad \{ \text{ right linearity of join (2.20); converse } \} \\
& (c \cdot \pi_1)^\circ \cup (c \cdot \epsilon \cdot \pi_2)^\circ \\
\Box &
\end{array}
$$

Proof of (B.17):

$$
\begin{array}{ll}
& \phi_p \cdot R \\
= & \quad \{ \text{ definition of } \phi_p \ \} \\
& (id \cap \dfrac{true}{p}) \cdot R \\
= & \quad \{ \text{ (2.25) because } \frac{true}{p} \cdot \mathrm{img}\, R \subseteq \frac{true}{p} \ \} \\
& R \cap \dfrac{true}{p} \cdot R \\
= & \quad \{ \text{ symmetric division (2.112) } \} \\
& R \cap p^\circ \cdot true \cdot R \\
= & \quad \{ \ true \cdot R = true \text{ if } R \text{ is reflexive; symmetric division (2.112) } \} \\
& R \cap \dfrac{true}{p} \\
\Box &
\end{array}
$$

B.4 THINNING

Theorem 5.2 (Theorem 9.2 from Bird and de Moor (1997)). With h monotonic with respect to R and such that $h \cdot \mathbb{F}\, H \cdot Q^\circ \ \subseteq \ R^\circ \cdot h \cdot \mathbb{F}\, H$ holds, we reason:

$$
\begin{array}{ll}
& \langle \mu\, X \ :: \ (h \cdot \in \restriction R) \cdot \mathbb{P}\, \mathbb{F}\, X \cdot (T^\circ \lfloor Q) \rangle \subseteq M \\
\Leftarrow & \quad \{ \text{ least prefix point } \} \\
& (h \cdot \in \restriction R) \cdot \mathbb{P}\, \mathbb{F}\, M \cdot (T^\circ \lfloor Q) \subseteq M \\
\equiv & \quad \{ \text{ universal property of shrinking (2.185) } \} \\
& \left\{ \begin{array}{l} (h \cdot \in \restriction R) \cdot \mathbb{P}\, \mathbb{F}\, M \cdot (T^\circ \lfloor Q) \subseteq H \\ (h \cdot \in \restriction R) \cdot \mathbb{P}\, \mathbb{F}\, M \cdot (T^\circ \lfloor Q) \cdot H^\circ \subseteq R \end{array} \right.
\end{array}
$$

First condition:

$$
\begin{array}{ll}
& (h \cdot \in \restriction R) \cdot \mathbb{P}\, \mathbb{F}\, M \cdot (T^\circ \lfloor Q) \subseteq H \\
\Leftarrow & \quad \{ \text{ shrinking cancellation (2.187) } \} \\
& h \cdot \in \cdot \mathbb{P}\, \mathbb{F}\, M \cdot (T^\circ \lfloor Q) \subseteq H \\
\Leftarrow & \quad \{ \ \mathbb{P}\, R \subseteq \in \setminus R \cdot \in \text{ (2.148) } \} \\
& h \cdot \in \cdot (\in \setminus \mathbb{F}\, M \cdot \in) \cdot (T^\circ \lfloor Q) \subseteq H \\
\Leftarrow & \quad \{ \text{ division cancellation (2.93) } \} \\
& h \cdot \mathbb{F}\, M \cdot \in \cdot (T^\circ \lfloor Q) \subseteq H
\end{array}
$$

$\Leftarrow$ { thinning cancellation (5.4) }

$h \cdot \mathbb{F}\ M \cdot T^\circ \subseteq H$

$\Leftarrow$ { shrinking cancellation (2.187) }

$h \cdot \mathbb{F}\ H \cdot T^\circ \subseteq H$

$\equiv$ { least prefix point of hylomorphism (2.219) }

$true$

□

Second condition:

$(h \cdot \in \upharpoonright R) \cdot \mathbb{P}\ \mathbb{F}\ M \cdot (T^\circ \downharpoonright Q) \cdot H^\circ \subseteq R$

$\Leftarrow$ { hylomorphism least fixpoint (2.220); converse }

$(h \cdot \in \upharpoonright R) \cdot \mathbb{P}\ \mathbb{F}\ M \cdot (T^\circ \downharpoonright Q) \cdot T \cdot \mathbb{F}\ H^\circ \cdot h^\circ \subseteq R$

$\Leftarrow$ { thinning cancellation (5.5) }

$(h \cdot \in \upharpoonright R) \cdot \mathbb{P}\ \mathbb{F}\ M \cdot \in^\circ \cdot Q \cdot \mathbb{F}\ H^\circ \cdot h^\circ \subseteq R$

$\equiv$ { converse }

$(h \cdot \in \upharpoonright R) \cdot \mathbb{P}\ \mathbb{F}\ M \cdot \in^\circ \cdot (h \cdot \mathbb{F}\ H \cdot Q^\circ)^\circ \subseteq R$

$\Leftarrow$ { assumption: $h \cdot \mathbb{F}\ H \cdot Q^\circ \subseteq R^\circ \cdot h \cdot \mathbb{F}\ H$ }

$(h \cdot \in \upharpoonright R) \cdot \mathbb{P}\ \mathbb{F}\ M \cdot \in^\circ \cdot (R^\circ \cdot h \cdot \mathbb{F}\ H)^\circ \subseteq R$

$\Leftarrow$ { $\mathbb{P}\ R \cdot \in^\circ \subseteq \in^\circ \cdot R$ (2.155) }

$(h \cdot \in \upharpoonright R) \cdot \in^\circ \cdot \mathbb{F}\ M \cdot (R^\circ \cdot h \cdot \mathbb{F}\ H)^\circ \subseteq R$

$\equiv$ { converse; function shrinking (2.192) }

$h \cdot (\in \upharpoonright R_h) \cdot \in^\circ \cdot \mathbb{F}\ M \cdot \mathbb{F}\ H^\circ \cdot h^\circ \cdot R \subseteq R$

$\Leftarrow$ { functor laws; shrinking cancellation (2.188) (twice) }

$h \cdot R_h \cdot \mathbb{F}\ R \cdot h^\circ \cdot R \subseteq R$

$\equiv$ { $R_h = h^\circ \cdot R \cdot h$ (2.6) }

$h \cdot h^\circ \cdot R \cdot h \cdot \mathbb{F}\ R \cdot h^\circ \cdot R \subseteq R$

$\Leftarrow$ { h is monotonic on R (2.132) }

$h \cdot h^\circ \cdot R \cdot R \cdot h \cdot h^\circ \cdot R \subseteq R$

$\equiv$ { h is simple (2.41) (twice); R is transitive (2.37) }

$R \cdot R \subseteq R$

$\equiv$ { R is transitive (2.37) }

$true$

□

Proof of (5.16):

$$
\begin{array}{ll}
& \mu \cdot \mathbb{P}\ (\in \mathbin{\rfloor} R) \subseteq (\in \mathbin{\rfloor} R) \cdot \mu \\
\equiv & \qquad \{\ \text{shunting (2.59); universal property of thinning (5.2)}\ \} \\
& \left\{ \begin{array}{l} \in \cdot\, \mu \cdot \mathbb{P}\ (\in \mathbin{\rfloor} R) \cdot \mu^\circ \subseteq \in \\ \mu \cdot \mathbb{P}\ (\in \mathbin{\rfloor} R) \cdot \mu^\circ \cdot \in^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\equiv & \qquad \{\ \text{converse; } \in \cdot\, \mu = \in \cdot\, \in \text{ (2.158)}\ \} \\
& \left\{ \begin{array}{l} \in \cdot\, \in \cdot\, \mathbb{P}\ (\in \mathbin{\rfloor} R) \cdot \mu^\circ \subseteq \in \\ \mu \cdot \mathbb{P}\ (\in \mathbin{\rfloor} R) \cdot \in^\circ \cdot \in^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\Leftarrow & \qquad \{\ \text{free theorem of } \in \text{ (2.154); (2.155)}\ \} \\
& \left\{ \begin{array}{l} \in \cdot\, (\in \mathbin{\rfloor} R) \cdot \in \cdot\, \mu^\circ \subseteq \in \\ \mu \cdot \in^\circ \cdot (\in \mathbin{\rfloor} R) \cdot \in^\circ \subseteq \in^\circ \cdot R \end{array} \right. \\
\Leftarrow & \qquad \{\ \text{thinning cancellation (5.4, 5.5)}\ \} \\
& \left\{ \begin{array}{l} \in \cdot\, \in \cdot\, \mu^\circ \subseteq \in \\ \mu \cdot \in^\circ \cdot \in^\circ \cdot R \subseteq \in^\circ \cdot R \end{array} \right. \\
\Leftarrow & \qquad \{\ \text{monotonicity (2.27); shunting (2.58, 2.59) (twice); converses; } \in \cdot\, \mu = \in \cdot\, \in \text{ (2.158)}\ \} \\
& \left\{ \begin{array}{l} \in \cdot\, \in \subseteq \in \cdot\, \in \\ \in \cdot\, \in \subseteq \in \cdot\, \in \end{array} \right. \\
\Box &
\end{array}
$$

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.