A Class of Graph Algorithms

Jesús N. Ravelo
Linacre College, Oxford

January 1996

Submitted in support of an application for transfer to D.Phil. status.

Oxford University Computing Laboratory
Programming Research Group

A Class of Graph Algorithms

Jesús N. Ravelo

Abstract

Several graph problems are shown to be instances of the abstract problem of selecting representative elements from a closure-generated set. These are the single-source minimum paths problem, the reachability set of a given vertex, and the construction of rooted trees. Algorithms to compute such representatives are derived from the specification of the general problem. Instances of the algorithms include Dijkstra's minimum paths algorithm, and depth-first/breadth-first graph traversals. A relational calculus for algorithms derivation is used as algebraic framework. This calculus, of a point-free reasoning character, is extended with categorical points to allow a clearer phrasing and manipulation of some algorithmic expressions. Pointwise reasoning can thus be naturally carried out within the extended calculus.

Contents

1 Introduction

Over the last two decades the calculational approach to algorithm design and software construction has become an important area of research in computer science. A formal specification of a problem is taken as the starting point, and transformation rules, guaranteed to preserve semantics, are then successively applied until an algorithm is obtained. The final product is certain to meet its formally stated requirements, and the key insights into the design of the algorithm can be better appreciated. Herein we specify a class of graph problems and derive algorithmic solutions for them. The class of problems include the single-source minimum paths problem, the reachability set of a given vertex, and the construction of rooted trees. They are treated uniformly as instances of an abstract specification. Generic algorithms are obtained, and instantiated to Dijkstra's minimum paths algorithm, depth-first and breadth-first traversals. The algebraic framework for algorithms derivation used is a categorical calculus of relations developed by Bird and de Moor [13]. This calculus, of a point-free reasoning character, is extended with categorical points to allow a clearer phrasing of some algorithmic expressions that often arise. Pointwise reasoning can thus be naturally carried out within the extended calculus.

A different presentation of some of the results given here can be found in [9]. Therein, an imperative algorithm is presented and its correctness proved. We now look for derived (instead of proved-correct) functional (instead of imperative) algorithms. Formalisation of the points extension of the calculus was the key to a succesful development. The same class of graph problems has also been tackled by other computer scientists using different calculational frameworks [5, 38, 42]. There are two main features brought together in this work. On one hand, a variety of algorithms linked to the well-developed subject of graph theory and, on the other hand, new methodologies for algorithm design characterised by their emphasis on calculation. A brief account of these subjects follow.

1.1 Graph Algorithmics

Graph theory is nowadays an important tool in the solution of a wide variety of practical problems in many different fields. Much of the theory has actually been developed motivated by its practical applications. This can be seen in what is recognised as the origin of graph theory, Euler's solution in 1735 of the Königsberg bridge puzzle, and several further developments such as Kirchhoff's theory of trees in 1847 for its application in the study of electrical networks and Ford and Fulkerson's theory of network flows in 1956 as an application in operations research. Other fields that have benefited from graph theoretical results include chemistry, biology, economics, geography, architecture, and social sciences (see e.g. [22]). In most practical situations that arise in these fields, graphs that model a real-life problem must be analysed in some way or other, and such graphs tend to be large and complex. Computer assistance to perform those analyses is then most valuable and, consequently, so is the design of efficient algorithms. A large number of graph algorithms have indeed been developed over the last few decades, and a class of graph problems for which efficient algorithms are not likely to be found has been identified, giving birth to the theory of NP-Completeness. A good introduction to graph theory and algorithmics can be found in [47].

Clearly, graph algorithmics is a most relevant subject with applications in many other fields. Most algorithmics textbooks dedicate significant space to its study (see e.g. [16, 29, 48]). However, their presentations follow, more often than not, a traditional approach to algorithm design. Algorithms are first given and afterwards, if at all, shown correct with respect to their specification. Several well-known graph algorithms, though sometimes compactly presented, have complicated correctness proofs. Authors must take pains to present clear proofs and yet, due to the lack of adequate algebras to do so, their efforts often result in arguments that are still difficult to read. In a different style (and serving other purposes), Knuth presents graph "literate programs" in [30] to be both read by humans and interpreted by machines. The presentation parts intended for humans are dedicated to explain features of the algorithms in their final form, with little insight to the design decisions that originated them.

Our main goal is to reveal the core of the strategies used to design algorithms in an algebraic fashion. There is a large variety of well-known graph algorithms based on several different paradigms and equally diverse graph theoretical concepts. Some graph algorithms are instances of general design paradigms. Kruskal's algorithm and Prim's algorithm for minimum spanning trees [31, 41], and Dijkstra's minimum paths algorithm [19] are examples of the greedy strategy. Some others are designed in very specific and peculiar ways, yet based on similar structural graph properties. This is the case of a class of algorithms based on depth-first search traversals, including a version of topological sorting, Hopcroft and Tarjan's algorithm and Kosaraju and Sharir's algorithm for strongly connected components [25, 44], Tarjan's biconnected components [46], and many others. Problems of these two classes are worth analysing in an algebraic framework. The former for their common design patterns, and the latter for their similar graph structural grounds. The effectiveness of such analyses depends on an adequate abstract setting for the study of both program construction and graphs.

1.2 Program Construction Framework

As mentioned earlier, we use a categorical calculus of relations for program construction. Specifications of problems for which algorithmic solutions are to be sought are naturally non-deterministic, associating input data to desired outputs in a relational fashion. Deterministic algorithms that "solve" a specification are just functions that respect such relation, i.e. that are included (in a set-theoretical sense) in it. A calculus of relations permits algorithms and specifications to be inmersed in the same reasoning space. The importance of relations to software development is already present in Dijkstra's seminal work on predicate transformers for imperative programs [20]. It has evolved into a uniform setting for specifications and programs in Morgan's refinement calculus [39]. A more general framework is needed if we also want to make space for functional programs. Indeed, Bird and de Moor's (BdM's) relational calculus developed from work on a calculus for functional programs by Bird and Meertens [8, 33]. The relevance of functional languages is nowadays largely recognised (see e.g. [26, 27]) after Backus convincingly argued in favour of them in the late 70s [6].

A typical case of indeterminacy is found in optimization problems, because several different solutions can all be optimal. This was one of the motivations for de Moor's initial work on relational derivation of programs [18]. When it comes to graph problems, optimization is found all over the place: minimum spanning trees, maximum matchings, shortest paths, maximum network flows, etc. Generic algorithmic solutions for several classes of optimisation problems, including some of the aforementioned graph problems, have been worked out within BdM's calculus [10, 11, 12, 13, 18]. Nevertheless, a number of them remain uncovered, which suggests more research to be undertaken with an eye to graph algorithms.

From a different perspective, the nature of graphs is relational, which makes a relational framework most convenient to reason about them. However, there is a subtle conflict in this dual advantage of a calculus of relations to derive algorithms and manipulate graphs. BdM's calculus treat specifications/algorithms in a point-free manner, thereby abstracting from data in order to reason more compactly about the processing of it. And graphs are the data to be processed. This will not actually represent a problem, but we need bear in mind the dual role the calculus can play analysing algorithms and graph properties. Schmidt and Ströhlein have exploited the applicability of relational calculi to graphs manipulation. In [43] they present, in relation-algebraic terms, a fair amount of graph theoretical concepts along with applications to databases, games and program verification. More concerned with algorithm derivation, Möller has developed a calculus of formal languages and relations with which compact calculations of some graph and pointer algorithms have been successfully carried out [37, 38, 42]. In his framework, relations are only used to reason about data, while algorithms are manipulated at a functional level. Indeterminacy and partiality are dealt with via set-valued functions, so a relational treatment of algorithms is present in spirit yet not explicitly.

The other important feature of BdM's calculus is its categorical foundation. Compactness and concision are most valuable in the program transformation area. Excessive detail in the expressions being manipulated makes them long and complex, thereby obscuring argumentations, often unnecessarily. Category theory advocates minimisation of such detail, providing an adequate and well-developed framework to build compact mathematical arguments. It had gained a great deal of interest from the computer science community in recent years [7, 40], in particular from the functional programming advocates. Interestingly enough, the laws of the calculus of functions of Bird and Meertens were later found to be governed by more general categorical concepts [45]. Freyd and Ščedrov's allegories [24] extends category theory, initially concerned with the motivating model of functions, to relations; BdM's calculus builds on this work. Also, a categorical treatment of datatypes has benefited the program transformation area [32], and the approach is extendable to the relational, or allegorical, setting (see [18]). Backhouse and others at Eindhoven University have been exploring the same idea of relation-algebraic treatment of datatypes [2], yet not relying on categorical grounds. The level of abstraction is high enough to allow concise reasoning about programs parameterised by datatypes; two examples can be found in [4, 14].

1.3 Outline

The present chapter 1 introduces the goals of this work, and related computer science areas. Chapter 2 briefly presents the main features of BdM's calculus, to proceed with the specification of the class of problems in chapter 3. The relational calculus is extended to include categorical pointwise reasoning in chapter 4, and such extension is immediately applied to the manipulation of part of the generic problem at issue. Chapter 5 contains the main derivation of the graph algorithms. Concluding remarks are given in the final chapter 6.

2 A Point-Free Calculus of Relations

The basics of the calculus of relations of Bird and de Moor, mainly based on axioms drawn from allegory theory [24], are introduced in this chapter. More than what is contained here will be needed, so other features will be reviewed in coming chapters as we proceed. Knowledge of categories and functors is assumed.

Basic Structure We work in the category $\mathrm{R{\scriptsize EL}}$ of sets and relations. Relations are typed backwards, denoting by $R : X \leftarrow Y$ a relation to X from Y; we think of relations as taking inputs on the right and delivering outputs on the left. Consistently, composition goes backwards, with $R \cdot S : X \leftarrow Z$ for $R : X \leftarrow Y$ and $S : Y \leftarrow Z$. Composition is associative with identity relations $id : X \leftarrow X$ as units.

Each relation $R : X \leftarrow Y$ has a converse relation $R^{\circ} : Y \leftarrow X$; converse preserves identities and reverses composition (i.e. $(R \cdot S)^{\circ} = S^{\circ} \cdot R^{\circ}$). Each homset $\mathrm{R{\scriptsize EL}}(X,Y)$, that is, each set of relations of type $X \leftarrow Y$, forms a complete, completely distributive, complemented lattice $(\mathrm{R{\scriptsize EL}}(X,Y), \cup, \cap, \overline{}, \emptyset, \Pi)$. $\cup$ and $\cap$ denote the join and meet operators, $\overline{}$ the complementation or negation operator, and $\emptyset$ and Π are the empty (bottom) and universal (top) relations. Hence, relations can be compared via a partial order $\subseteq$, where $R \subseteq S$ denotes $R = R \cap S$, and also $S = R \cup S$. This ordering is preserved by converse.

Composition distributes over join, but only weakly distributes over meet in that

$$\begin{array}{rcl} (R \cap S) \cdot T & \subseteq & R \cdot T \ \cap \ S \cdot T \\ R \cdot (S \cap T) & \subseteq & R \cdot S \ \cap \ R \cdot T \end{array}$$

Composition is assumed to bind more tightly than join and meet. Another axiom, called the modular law, gives a weak converse of distributivity over meet:

$$\begin{array}{rcl} R \cdot S \ \cap \ T & \subseteq & R \cdot (S \ \cap \ R^{\circ} \cdot T) \\ R \cdot S \ \cap \ T & \subseteq & (R \ \cap \ T \cdot S^{\circ}) \cdot S \end{array}$$

Other Operators We shall use the notions of division, implication and substraction in the calculus. From distributivity of composition and meet over join, and the theory of Galois connections [1], division and implication can be uniquely characterised by

$$\begin{array}{rcl} X \subseteq R \backslash S & \equiv & R \cdot X \subseteq S \\ X \subseteq S / R & \equiv & X \cdot R \subseteq S \\ X \subseteq R \Rightarrow S & \equiv & X \cap R \subseteq S \end{array}$$

They can be interpreted set-theoretically as follows:

$$\begin{array}{rcl} y\,(R\backslash S)\,x & \equiv & (\forall z : z\,(R)\,y : z\,(S)\,x) \\ y\,(S/R)\,x & \equiv & (\forall z : x\,(R)\,z : y\,(S)\,z) \\ y\,(R \Rightarrow S)\,x & \equiv & (y\,(R)\,x \;\Rightarrow\; y\,(S)\,x) \end{array}$$

Equally, substraction can be defined by

$$R - S \subseteq X \quad \equiv \quad R \subseteq X \cup S$$

In the presence of complementation, implication and substraction satisfy respectively $R \Rightarrow S = \overline{R} \cup S$ and $R - S = R \cap \overline{S}$. The distribution properties of substraction will be used a good deal:

$$\begin{array}{rcl} (R_1 \cup R_2) - S & = & (R_1 - S) \cup (R_2 - S) \\ R - (S_1 \cup S_2) & = & R - S_1 - S_2 \end{array}$$

Functions A relation R is entire if $id \subseteq R^\circ \cdot R$, and simple if $R \cdot R^\circ \subseteq id$. That is, an entire relation delivers at least one output for each input, and a simple one delivers at most one output for each input. A function is an entire and simple relation. Sets and functions form a subcategory FUN of REL. Functions will be usually denoted by the letters f, g, h. Useful properties of functions include the so-called shunting rules

$$\begin{array}{rcl} f \cdot R \subseteq S & \equiv & R \subseteq f^\circ \cdot S \\ R \cdot f^\circ \subseteq S & \equiv & R \subseteq S \cdot f \end{array}$$

and that inclusion of functions is equivalent to equality:

$$f \subseteq g \quad \equiv \quad f = g$$

Least Fixed Points Since the homsets of relations are complete lattices, monotonic endofunctions ϕ on them (i.e., such that $X_1 \subseteq X_2 \Rightarrow \phi X_1 \subseteq \phi X_2$) have least fixed points. The well-known Knaster-Tarski theorem states that such a least fixed point is also the least solution of $\phi X \subseteq X$. We denote it by $(\mu X : \phi X)$, and characterise it by the two aforementioned properties:

$$\begin{array}{l} (\mu X : \phi X) = \phi(\mu X : \phi X) \\ (\mu X : \phi X) \subseteq Y \quad \Leftarrow \quad \phi Y \subseteq Y \end{array}$$

Closure A preorder R is a relation which is both reflexive ($id \subseteq R$) and transitive ($R \cdot R \subseteq R$). The reflexive-transitive closure S^* of a relation S is the least preorder that contains S. It is a well-known fact that S^* is also the least solution of other equations:

$$\begin{aligned} S^* &= (\mu X : id \cup X \cdot S) \\ &= (\mu X : id \cup S \cdot X) \end{aligned}$$

The second one is a particular case of the more general result

$$S^* \cdot T = (\mu X : T \cup S \cdot X)$$

Finally, note that reflexivity and transitivity are preserved by converse. Thus, the converse of a preorder is also a preorder.

Powersets We can move between a relation $R : X \leftarrow Y$ and its corresponding set-valued function $\Lambda R : \mathsf{P}X \leftarrow Y$, called the power-transpose of R and defined in set theory as $(\Lambda R)\, y = \{\, x \mid x\,(R)\,y \,\}$. This isomorphism is formalised in the calculus as follows. For every set X there exists a set $\mathsf{P}X$, called the powerset of X, and a relation $\in\, : X \leftarrow \mathsf{P}X$, called the membership relation on X. For every relation $R : X \leftarrow Y$, there exists a function $\Lambda R : \mathsf{P}X \leftarrow Y$ such that

$$f = \Lambda R \quad \equiv \quad \in \cdot f = R \qquad\qquad \text{for all functions } f : \mathsf{P}X \leftarrow Y$$

From the above it follows that power-transpose absorbs functions, that is, for every R relation and f function $\Lambda(R \cdot f) = \Lambda R \cdot f$.

Products and Coproducts For every pair of sets X and Y, we can form its product $X \times Y$ and its sum or coproduct $X + Y$. Products come equipped with projection functions *outl* and *outr*, and for each pair of relations $R : X \leftarrow Z$ and $S : Y \leftarrow Z$, their "split" $\langle R, S \rangle : X \times Y \leftarrow Z$ is defined as

$$\langle R, S \rangle = outl^\circ \cdot R \cap outr^\circ \cdot S$$

Analogously, coproducts are equipped with injection functions *inl* and *inr*, and for every $R : Z \leftarrow X$ and $S : Z \leftarrow Y$, the "case" relation $[R, S] : Z \leftarrow X + Y$ is defined as

$$[R, S] = R \cdot inl^\circ \cup S \cdot inr^\circ$$

Product and coproduct become functors in REL defining

$$\begin{aligned} R \times S &= \langle\, R \cdot outl\,,\, S \cdot outr \,\rangle \\ R + S &= [\, inl \cdot R\,,\, inr \cdot S \,]. \end{aligned}$$

3 A Class of Graph Problems

This chapter presents the specification of the class of graph problems mentioned in the introduction. We start with the richest one, the minimum paths problem, and then proceed with the others. At the end, an abstract specification that unifies them all is presented. Before proceeding with the problems, graphs are introduced.

A graph G is a 4-tuple $(V, E, to, from)$, where V and E are respectively the set of vertices and the set of edges, and *to* and *from* are functions of type $V \leftarrow E$ that determine the extreme vertices of an edge. As expected, an edge e goes to $to\,e$ from $from\,e$. Note that both loops and multiple edges are allowed. Given a graph G, its adjacency relation $vadj : V \leftarrow V$ between vertices is defined as $vadj = to \cdot from^{\circ}$. Although less commonly used, we can also define a adjacency relation $eadj : E \leftarrow E$ between edges as $eadj = from^{\circ} \cdot to$.

3.1 Single-Source Minimum Paths

In this problem we are provided with a weighted graph, that is, a graph G and a function $weight : \mathbb{R}^{+} \leftarrow E$ that assigns a weight, or cost, to each edge. We are also given a vertex s, called the source. For every vertex v reachable from s, a minimum path to v from s is to be found. We aim to specify this problem in the form

$$minpaths \quad = \quad selectmins \cdot genallpaths$$

The generating function *genallpaths* takes a vertex s and returns the set of all paths in G that start in s. The other component *selectmins* gives a subset of its input that contains a minimum path for each ending vertex; it is a relation because minima are not unique.

We first proceed with the generator. A path to v from u in a graph G is a sequence

$$\begin{array}{ll} \multicolumn{2}{l}{[v_n, e_n, v_{n-1}, e_{n-1}, \ \ldots \ v_1, e_1, v_0],} \\ \text{where} & (\forall i : 1 \leq i \leq n : \ to\ e_i = v_i \ \text{ and } \ from\ e_i = v_{i-1}) \\ \text{and} & v = v_n \\ \text{and} & u = v_0 . \end{array}$$

Clearly, there is some redundancy in this definition of paths. For a non-empty path (i.e. one in which $n \geq 1$), all the information can be recovered from just its subsequence of edges. This is not the case for empty paths, all of which have an empty subsequence of edges. In order to avoid unnecessary redundancy without loss of information, we shall represent a path by its subsequence of edges and the initial vertex $[e_n, e_{n-1}, \ \ldots \ e_1, v_0]$. Symmetrically, the final vertex and edges would do, but the chosen representation suits better the problem in analysis.[1] Thus, using Orwell-like notation [36] the type *Path* is defined as

$$Path \quad ::= \quad wrap\ V \ \mid \ cons\ (E, Path).$$

The set-valued function *genallpaths* can be specified as the power-transpose of the relation $pathfrom : Path \leftarrow V$ such that $p\,(pathfrom)\,v$ if p is a path starting in v. We define it as $pathfrom = ispath? \cdot start^{\circ}$ where *start* is the function

$$\begin{aligned} start\,(wrap\,v) &= v \\ start\,(cons\,(e,p)) &= start\,p \end{aligned}$$

and *ispath* is the predicate

$$\begin{aligned} ispath\,(wrap\,v) &= True \\ ispath\,(cons\,(e,p)) &= ispath\,p \ \wedge \ from\,e = end\,p \end{aligned}$$

Predicates are modelled by coreflexive relations, i.e., subsets of the identity relation. $ispath?$ above is such that $p\,(ispath?)\,p$ if and only if predicate *ispath* holds for p. The function *end* is the counterpart to *start* that gives the final vertex of a path:

$$\begin{aligned} end\,(wrap\,v) &= v \\ end\,(cons\,(e,p)) &= to\,e \end{aligned}$$

Having done with the generator *genallpaths*, we now turn to *selectmins*. A minimum path for each ending vertex is wanted. Hence, we can think of partitioning the set of generated paths according to their ends, and then drawing a minimum from each of such classes. This idea can be generalised to a parameterised selector $mins(Q, R)$, with Q an equivalence relation and R a preorder. (An equivalence relation is a symmetric ($Q^{\circ} = Q$) preorder.) The *mins* relation is typed $\mathsf{P}X \leftarrow \mathsf{P}X$ for Q and R of type $X \leftarrow X$; and $x\,(mins)\,y$ if x contains, for each class in y induced by Q, one (and only one) minimum element according to R. Its formal definition is postponed to the end of this chapter. For our specific problem, two paths will be regarded as equivalent if they end in the same vertex:

$$sameends \quad = \quad end^{\circ} \cdot end.$$

[1]The mirror version, single-destination minimum paths, would benefit from the final vertex representation of paths. Nevertheless, each version of the problem can be expressed in terms of the other by means of the converse graph $conv\ G = (V, E, from, to)$.

The cost of a path is defined to be the sum of the weights of its edges. Via such cost function, a preorder *better* over paths is induced by the standard ordering over numbers:

$$
\begin{array}{rcl}
cost\,(wrap\,v) & = & 0 \\
cost\,(cons\,(e,p)) & = & weight\,e \;+\; cost\,p \\
better & = & cost^{\circ} \cdot leq \cdot cost
\end{array}
$$

where the symbol $+$ above refers to addition of real numbers and *leq* is the less-than-or-equal relation.

The specification for the single-source minimum paths problem then reads

$$
minpaths \quad = \quad mins\,(sameends, better) \cdot \Lambda pathfrom
$$

Transforming the Generator

We now transform the generator into a more suitable form for later manipulations. For that, we need to review the categorical approach to datatypes. (A good introduction can be found in [32], more complete relational treatments in [2, 13].) The above declaration of the datatype *Path* declares $[wrap, cons] : Path \leftarrow \mathsf{F}Path$ to be the initial F-algebra of the functor $\mathsf{F}X = V + E \times X$. This means that for every other F-algebra $[R, S] : X \leftarrow \mathsf{F}X$ there is a unique homomorphism $(\![R, S]\!) : X \leftarrow Path$, called a catamorphism [34], characterised by

$$
T = (\![R, S]\!) \quad \equiv \quad T \cdot [wrap, cons] = [R, S] \cdot \mathsf{F}T.
$$

The right-hand side above can be read

$$
\begin{array}{rcl}
T \cdot wrap & = & R \\
T \cdot cons & = & S \cdot (id \times T)
\end{array}
$$

With this machinery, we can define *start*, *end*, *ispath*? and *cost* as follows:

$$
\begin{array}{rcl}
start & = & (\![\, id \,,\, outr \,]\!) \\
end & = & (\![\, id \,,\, to \cdot outl \,]\!) \\
ispath? & = & (\![\, wrap \,,\, cons \cdot step\,(from^{\circ} \cdot end) \,]\!) \\
cost & = & (\![\, zero! \,,\, plus \cdot (weight \times id) \,]\!)
\end{array}
$$

The coreflexive $step\,R$ corresponds to a predicate on pairs that holds on (x, y) if $x\,(R)\,y$. It is defined as $step\,R = id \cap outl^{\circ} \cdot R \cdot outr$.

Our generator has the form of a hylomorphism [35], that is, the composition of a catamorphism after the converse of another. Hylomorphisms can be expressed as least fixed points: $(\![R]\!) \cdot (\![S]\!)^\circ = (\mu X : R \cdot \mathsf{F}X \cdot S^\circ)$. This will allow us to transform *pathfrom* into a closure expresion:

$$
\begin{array}{cl}
 & \mathit{pathfrom} \\
= & \quad \{ \text{ definition and hylomorphisms } \} \\
 & (\mu X : [\, \mathit{wrap} \,,\, \mathit{cons} \cdot \mathit{step}\,(\mathit{from}^\circ \cdot \mathit{end})\,] \cdot (\mathit{id} + \mathit{id} \times X) \cdot [\, \mathit{id} \,,\, \mathit{outr}\,]^\circ) \\
= & \quad \{ \text{ coproducts } \} \\
 & (\mu X : \mathit{wrap} \;\cup\; \mathit{cons} \cdot \mathit{step}\,(\mathit{from}^\circ \cdot \mathit{end}) \cdot (\mathit{id} \times X) \cdot \mathit{outr}^\circ) \\
= & \quad \{ \text{ products } \} \\
 & (\mu X : \mathit{wrap} \;\cup\; \mathit{cons} \cdot \mathit{step}\,(\mathit{from}^\circ \cdot \mathit{end}) \cdot \mathit{outr}^\circ \cdot X) \\
= & \quad \{ \; \mathit{step} \text{ property } \} \\
 & (\mu X : \mathit{wrap} \;\cup\; \mathit{cons} \cdot \langle\, \mathit{from}^\circ \cdot \mathit{end} \,,\, \mathit{id} \,\rangle \cdot X) \\
= & \quad \{ \text{ closure, introducing } \mathit{extend} \; \} \\
 & \mathit{extend}^* \cdot \mathit{wrap}
\end{array}
$$

Hence, the generator takes the following shape using the fact that *wrap* is a function and the power-transpose absorption law:

$$
\begin{array}{l}
\Lambda \mathit{pathfrom} \;=\; \Lambda(\mathit{extend}^* \cdot \mathit{wrap}) \;=\; \Lambda \mathit{extend}^* \cdot \mathit{wrap} \\
\text{where} \\
\qquad\qquad \mathit{extend} \;=\; \mathit{cons} \cdot \langle\, \mathit{from}^\circ \cdot \mathit{end} \,,\, \mathit{id} \,\rangle.
\end{array}
$$

3.2 Single-Source Shortest Paths – Unweighted Case

A simpler version of the single-source minimum paths problem is obtained by uniformly assigning a weight of one to each edge. The presence of multiple edges can thus be considered irrelevant, since any of them would contribute equally to the total cost. Assuming there are no multiple edges in a graph, the information provided by its adjacency relation is complete and so we can do only with (V, vadj) for the graph $G = (V, E, \mathit{to}, \mathit{from})$. That is, $(E, \mathit{to}, \mathit{from})$ can be uniquely determined, up to isomorphism, from (V, vadj) in this case. In allegorical terms, the functions *to* and *from* tabulate the relation *vadj*. A pair of functions f, g is a tabulation of R if $R = f \cdot g^\circ$ and f, g are jointly monic, i.e., $f^\circ \cdot f \cap g^\circ \cdot g = \mathit{id}$. The fact that the pair of functions involved in a tabulation are jointly monic correspond to the absence of parallel, or multiple, edges.[2]

We now proceed as in the last section to specify this variant of the paths problem. We do so only by means of (V, vadj), ignoring the edges information. A path is now a non-empty sequence of vertices $[v_n, \; \ldots \; v_1, v_0]$, such that

[2] This graph representation issue is also discussed in [43] as the correspondence between, under their terminology, directed graphs and 1-graphs. The discussion is slightly more general there since edges can lack either extreme –or both–; that is, *to* and *from* are only required to be simple.

$(\forall i : 1 \leq i \leq n : v_i \; vadj \; v_{i-1})$. The type *Path*, the functions *start* and *end*, and the "healthiness" coreflexive are redefined accordingly:

$$\begin{array}{rcl} Path & ::= & wrap\ V \ \mid\ cons\ (V, Path) \\ start & = & (\![\, id \,,\, outr \,]\!) \\ end & = & (\![\, id \,,\, outl \,]\!) \\ ispath? & = & (\![\, wrap \,,\, cons \cdot step\,(vadj \cdot end) \,]\!) \end{array}$$

The generator of all paths from the given source is again $\Lambda pathfrom$, with $pathfrom = ispath? \cdot start^{\circ}$. A manipulation similar to the one of the weighted case yields the desired closure:

$$\begin{array}{l} pathfrom \;=\; extend^{*} \cdot wrap \\ \text{where} \\ \qquad extend \;=\; cons \cdot \langle\, vadj \cdot end \,,\, id \,\rangle. \end{array}$$

Equivalence and cost ordering of the paths are easily adapted to

$$\begin{array}{rcl} sameends & = & end^{\circ} \cdot end \\ length & = & (\![\, zero! \,,\, add1 \cdot outr \,]\!) \\ shorter & = & length^{\circ} \cdot leq \cdot length \end{array}$$

with *add1* the successor function on natural numbers. We finally specify

$$\begin{array}{rcl} shpaths & = & mins\,(sameends, shorter) \cdot \Lambda pathfrom \\ & = & mins\,(sameends, shorter) \cdot \Lambda extend^{*} \cdot wrap \end{array}$$

3.3 Reachability

This is the simplest problem of all the ones considered herein. Its aim is to find all the vertices that can be reached from a given one. Such a set is trivially obtained using the (vertex) adjacency relation as base generator. As for the parameterisation of the selection, every vertex is relevant on its own so no pair of different elements are set to be equivalent. Using *id* as the equivalence relation for *mins* leaves the role of the comparison preorder useless, since each induced class is unitary; we shall use Π as a symbol for "no special preferences". This vacuous use of *mins* corresponds to the fact that for every R preorder $mins\,(id, R) = id$. The reachability set problem reads

$$reachset \;=\; mins\,(id, \Pi) \cdot \Lambda vadj^{*}$$

3.4 Rooted Trees

A rooted tree of a graph G is a subgraph G' of it with a distinguished vertex r (for *root*) such that: (i) every vertex is reachable from r, (ii) it has no circuits, and

(iii) every vertex has at most one predecessor. The problem we treat hereafter is the construction of full rooted trees of a given graph $G = (V, E, to, from)$, with the desired root $r \in V$ as input. We mean by a full tree one that contains every vertex reachable from the root.

We shall represent a tree by its set of edges, all the vertices being determined from it via the extreme functions *to* and *from* since a tree has no isolated nodes. Only when a tree is composed solely of its root do we have an isolated vertex (and an empty set of edges), such degenerate case will not represent a problem. We shall use a suitable base generator to initially obtain all the edges that can be reached from the given root. (Such full set of edges would comprise what is often called in the literature a rooted graph.) This full set will then be sift through an appropriate instantiation of *mins* to obtain a tree.

The obvious base generator is the edge adjacency relation. However, a starting edge is lacking since we are given not an edge but a node as starting-point. We tackle this problem defining an extended graph with an additional special vertex and edges leading from it to every other node, thereby creating a one-to-one correspondence between the original vertices and the added edges. Such correspondence provides us the starting edge associated to the input root. This "imaginary" edge will be part of any resulting tree and, therefore, so will be the added new vertex. The above-mentioned degenerate case of tree will then be made up of that single edge instead of none. A formal definition of the described extension for a graph $G = (V, E, to, from)$ follows (prefixes x, *orig* and *new* stand for extended, original and new, respectively):

$$
\begin{array}{l}
xgraph\ G \ = \ (xverts\ V, xedges\ E, xto, xfrom) \\
\text{where} \\
\qquad \begin{array}{lcl}
xverts\ V & ::= & origv\ V \mid newv \\
xedges\ E & ::= & orige\ E \mid newe\ V \\
xto & = & (\!|\, origv \cdot to\, ,\, origv \,|\!) \\
xfrom & = & (\!|\, origv \cdot from\, ,\, newv! \,|\!)
\end{array}
\end{array}
$$

The defined datatypes correspond to the coproducts $V + \mathbf{1}$ and $E + V$. The so-called unit of the allegory, $\mathbf{1}$, represents a one-element set.

We are now fully equipped to specify the problem. The relational generator is $xeadj^* \cdot newe$ where $xeadj = xfrom^\circ \cdot xto$. Its associated set-valued function generates the full subgraph reachable from the inputted root. This subgraph satisfies condition (i) of the definition of rooted tree. To comply with (ii) and (iii), some edges shall be filtered out. Assuming (i), the conjunction of conditions (ii) and (iii) is equivalent to

(ii') the root has no predecessors, and every other vertex has exactly one.

Since we made up a special predecessor for the root, this condition reads that every original vertex must have exactly one predecessor. The *mins* selector is used as a sieve setting two edges to be equivalent when they lead to the same node: $\mathit{sameends} = \mathit{xto}^\circ \cdot \mathit{xto}$. Among the edges of an equivalence class thereby constructed no one is preferred to any other and, therefore, the universal relation Π will serve as comparison preorder. However, condition (i) could be lost after the filtering. We use a coreflexive *connected*? after *mins* to ensure that condition (i) is preserved, i.e., that every edge can still be traced back within the set to the new special node. We specify

$$\mathit{rootedtree} \quad = \quad \mathit{connected}? \cdot \mathit{mins}\,(\mathit{sameends}, \Pi) \cdot \Lambda \mathit{xeadj}^* \cdot \mathit{newe}$$

3.5 Abstract Specification

A common pattern can be seen in the specifications shown before:

$$P \quad = \quad \mathit{mins}\,(Q, R) \cdot \Lambda S^*$$

We call P the minimum-representatives of closure problem. Q and R are an equivalence and a preorder that determine the selection of minima, and S is called the base generator or spawning relation. The problems of the sections above are instances of it, some with a particular initialisation, some with an additional post-condition. The next step is to refine P to algorithms that compute such representatives, which will be done in the following chapters. In the process, some conditions will be required on the parameters Q, R, S of the specification. Algorithmic solutions for instances of P can be obtained verifying that those provisos are met by the particular parameters.

For a formal definition of *mins*, recall that $x\,(\mathit{mins}(Q, R))\,y$ if x contains exactly one R-minimum for each Q-class in y. We define

$$\begin{array}{l} \mathit{mins}\,(Q, R) \;\widehat{=}\; \mathit{uniq}\ Q \cdot \mathit{thin}\ (Q \cap R) \\ \text{where} \\ \qquad \begin{array}{rcl} \mathit{uniq}\ S & = & \mathit{id}\ \cap\ \in\backslash(S \Rightarrow \mathit{id})/\ni \\ \mathit{thin}\ S & = & \in\backslash\in\ \cap\ (\ni\cdot S)/\ni \end{array} \end{array}$$

Informally, this says that a subset is returned with at most one element per Q-class such that every element of the input has a Q-equivalent but R-better element in the output. This corresponds to the previously stated informal notion of *mins*. For completeness sake, we spell out another definition of *mins*, somehow more natural in the sense that the notions of minimum and equivalence classes are directly used but that requires more calculus not introduced (the intersection or *cap* function and the powerset functor P):

$$\begin{array}{l} \mathit{mins}\,(Q, R) \;=\; \mathit{uniq}\ Q \cdot \mathsf{P}(\mathit{min}\ R) \cdot \Lambda(\mathit{class}\ Q) \\ \text{where} \\ \qquad \begin{array}{rcl} \mathit{min}\ R & = & \in\ \cap\ R/\ni \\ \mathit{class}\ Q & = & \mathit{cap} \cdot \langle\, \Lambda Q\cdot\in, \mathit{id}\,\rangle \end{array} \end{array}$$

Both definitions are equivalent provided Q is an equivalence relation and R a preorder. The first is more suitable for manipulations to be subsequently carried out.

4 Adding Pointwise Reasoning

Point-free reasoning has been shown to be effective in the area of program transformation. For that reason, among others, allegory theory has provided an adequate foundational framework for BdM's relational calculus. However, a pointwise proof is sometimes more effective, tabulations being the tool that has been used to carry out such proofs (see [13]). Curtis proposed an alternative way out through graphical representations, which often give hints of next steps in a proof more likely to succeed [17]. Yet another approach is to use categorical points, also called constants or elements: functions from the unit 1. Recall that 1 represents a unitary set.

Most of the cases in which point-free proofs are less effective that pointwise ones involve products and powersets. These two features sometimes make point-free expressions be pervaded by the combinators *outl*,*outr* and the membership relation $\in$ in such a way that the meaning of the whole tend to be obscure. In such cases proofs are often hard to work out, contradicting the symbol-guided nature that should characterise a point-free style. We propose to treat these cases extending BdM's calculus with the aforementioned notion of points. A systematic way to carry out such "categorical pointwise" proofs is shown in this chapter, which is then used to state and prove rules that manipulate the *mins* selector. These *mins* rules both show the applicability of this extension to BdM's and prepare the field for the later manipulation of the representatives of closure problem. It is worth mentioning that most graph problems seem to involve a good deal of the powerset and product combinators that motivated the introduction of points in the first place.

4.1 Using Points

The motivating model of allegory theory is that of sets and relations, yet many other models are accepted by the axioms of a "raw" allegory. Further assumptions, e.g. the existence of tabulations, narrow down the spectrum of models closer to set-theoretic relations. An important feature of relations, being defined as subsets of cartesian products in set theory, is that of being built up from (pairs of) elements. The corresponding notion to such elements in lattice theory is that of an atom, an immediate succesor of bottom. And a lattice is said to be atomic if every lattice member can be built up from atoms. We shall assume the allegory is atomic, meaning that each homset is an atomic lattice. Atoms take the form of a point after the converse of a point. We then have that every relation R satisfies

$$R \quad = \quad (\bigcup x, y \;:\; x, y \text{ points} \wedge x{\cdot}y^{\circ} \subseteq R \;:\; x{\cdot}y^{\circ})$$

Note that $x{\cdot}y^{\circ} \subseteq R$ corresponds to the intuitive appeals to set theory we have been doing of the form $x\,(R)\,y$. Also note that, by shunting of functions, it is equivalent to $x \subseteq R \cdot y$ which, under the right-to-left "operational" interpretation attached to relations, can be read as "x is a possible output of R for input y". We now count with the following alternatives to prove inclusion of relations:

$$\begin{aligned} R \subseteq S &\equiv (\forall\, y \,:\, y \text{ point} \,:\, R \cdot y \subseteq S \cdot y) \\ &\equiv (\forall\, x, y \,:\, x, y \text{ points} \,:\, x \subseteq R \cdot y \;\Rightarrow\; x \subseteq S \cdot y) \end{aligned}$$

Further rules follow. For example, to manipulate composition, functions and coreflexives, we have

$$\begin{aligned} x \subseteq R \cdot S \cdot y &\equiv (\exists\, z \,:\, z \text{ point} \,:\, x \subseteq R \cdot z \;\wedge\; z \subseteq S \cdot y) \\ x \subseteq f \cdot y &\equiv x = f \cdot y \\ x \subseteq C \cdot y &\equiv x = y \;\wedge\; x \subseteq C \cdot x \end{aligned}$$

Recall the modelling-predicates role of coreflexives. We shall denote by $C\,.\,x$ the satisfaction of x by C, meaning $x \subseteq C \cdot x$.

Everything else we shall need to manipulate relations through points will fall down naturally from the rules above and predicate calculus. As an example, we show that $R \cdot C \subseteq S$ can be proved by showing that $(C\,.\,x \Rightarrow R \cdot x \subseteq S \cdot x)$ for every point x:

$$\begin{aligned} & R \cdot C \subseteq S \\ \equiv \quad & \{ \text{ points-inclusion } \} \\ & (\forall y, x : y \subseteq R \cdot C \cdot x \;\Rightarrow\; y \subseteq S \cdot x) \\ \equiv \quad & \{ \text{ points-composition } \} \\ & (\forall y, x : (\exists z : y \subseteq R \cdot z \;\wedge\; z \subseteq C \cdot x) \;\Rightarrow\; y \subseteq S \cdot x) \\ \equiv \quad & \{ \text{ points-coreflexive, predicate calculus } \} \\ & (\forall y, x : y \subseteq R \cdot x \;\wedge\; C\,.\,x \;\Rightarrow\; y \subseteq S \cdot x) \\ \equiv \quad & \{ \text{ predicate calculus } \} \\ & (\forall x : C\,.\,x \;\Rightarrow\; (\forall y : y \subseteq R \cdot x \;\Rightarrow\; y \subseteq S \cdot x)) \\ \equiv \quad & \{ \text{ points-inclusion } \} \\ & (\forall x : C\,.\,x \;\Rightarrow\; R \cdot x \subseteq S \cdot x) \end{aligned}$$

Power-Products

It was mentioned that points seem to be particularly useful when powersets and products are involved. For powersets, due to the power-transpose isomorphism $\mathrm{FUN}(\mathsf{P}X, 1) \cong \mathrm{REL}(X, 1)$, elements of either homset represent points of $\mathsf{P}X$; uppercase and lowercase letters will be used to implicitly denote the correspondence $a = \Lambda A$. The same goes to the product isomorphism $\mathrm{FUN}(X \times Y, 1) \cong \mathrm{FUN}(X, 1) \times \mathrm{FUN}(Y, 1)$, with the convention $a = \langle a_1, a_2 \rangle$.

For later convenience, new notation is introduced to denote points of products of powerset types, e.g. $(\mathsf{P}X \times \mathsf{P}Y)$. We shall denote by $\langle\!| A, B |\!\rangle$ the relation $\langle \Lambda A, \Lambda B \rangle$. A kind of right-associativity will be assumed when more than two components are needed, this meaning that $\langle\!| A, B, C |\!\rangle$ denotes $\langle \Lambda A, \langle \Lambda B, \Lambda C \rangle\rangle$. Consistently, it will sometimes be used even for a single component, $\langle\!| A |\!\rangle$ meaning ΛA.

4.2 Mins Calculus

Recall the formal definition of *mins* given at the end of chapter 3:

$$
\begin{array}{l}
\mathit{mins}\,(Q,R) \;\widehat{=}\; \mathit{uniq}\ Q \cdot \mathit{thin}\ (Q \cap R) \\
\text{where} \\
\qquad \begin{array}{rcl}
\mathit{uniq}\ S &=& \mathit{id}\ \cap\ \in\backslash(S \Rightarrow \mathit{id})/\ni \\
\mathit{thin}\ S &=& \in\backslash\in\ \cap\ (\ni\cdot S)/\ni
\end{array}
\end{array}
$$

Let us show the pointwise reading of these relations. Bear in mind the naming convention whereby sets are modelled by relations $A : X \leftarrow 1$ that are in a one-to-one correspondence with powerset points $a = \Lambda A : \mathsf{P}X \leftarrow 1$. We manipulate in detail the *thin* case:

$$
\begin{array}{cl}
 & b \subseteq \mathit{thin}\, S \cdot a \\
\equiv & \quad \{ \text{ shunting of functions, definition of } \mathit{thin} \ \} \\
 & b \cdot a^\circ \ \subseteq\ \in\backslash\in\ \cap\ (\ni \cdot S)/\ni \\
\equiv & \quad \{ \text{ meet, division } \} \\
 & \in \cdot\, b \cdot a^\circ \subseteq \in \ \wedge\ b \cdot a^\circ \cdot \ni\ \subseteq\ \ni \cdot S \\
\equiv & \quad \{ \text{ shunting of functions } \} \\
 & \in \cdot\, b \subseteq \in \cdot\, a \ \wedge\ a^\circ \cdot \ni\ \subseteq\ b^\circ \cdot \ni \cdot S \\
\equiv & \quad \{ \text{ naming convention, power-transpose, converse } \} \\
 & B \subseteq A \ \wedge\ A \subseteq S^\circ \cdot B
\end{array}
$$

Analogously, we can also get the pointwise description of *uniq* and *mins*:

$$
\begin{array}{rcl}
b \subseteq \mathit{thin}\, S \cdot a &\equiv& B \subseteq A \ \wedge\ A \subseteq S^\circ \cdot B \\
\mathit{uniq}\, Q \,\bullet\, a &\equiv& A \cdot A^\circ \cap Q \subseteq \mathit{id} \\
b \subseteq \mathit{mins}\,(Q,R) \cdot a &\equiv& \left\{ \begin{array}{c} B \cdot B^\circ \cap Q \subseteq \mathit{id} \quad \wedge \\ B \subseteq A \ \wedge\ A \subseteq (Q \cap R)^\circ \cdot B \end{array} \right.
\end{array}
$$

Now for the rules.

Mins/Cup The function $\mathit{cup} = \Lambda(\in \cdot\, \mathit{outl}\ \cup\ \in \cdot\, \mathit{outr})$ returns the union of its two input sets. We aim to analyse the interaction of *mins* and *cup*. A way of choosing representatives from the union of two sets is to initially get them solely from the first one and then get the rest from the other only for classes not yet represented. It is safe to do so only if classes present in both sets have a minimum (of the union) in the first. A corresponding point-free statement, but with a stronger precondition, is:

$$
\begin{array}{l}
\mathit{mins} \cdot \mathit{cup} \quad \supseteq \quad \mathit{cup} \cdot \mathit{mins}^2 \cdot (\mathit{id} \times \mathit{drop}) \cdot \delta \cdot p? \\
\text{where} \\
\qquad \begin{array}{rcl}
\mathit{drop}\, Q & = & \mathit{diff} \cdot (\mathsf{E}Q \times \mathit{id}) \\
\mathit{diff} & = & \Lambda(\in \cdot\, \mathit{outr} \;-\; \in \cdot\, \mathit{outl}) \\
\delta & & \text{is the obvious natural} \\
 & & \text{transformation typed } \; X \times (X \times Y) \hookleftarrow X \times Y \\
p? & = & \mathit{step}\,(\in \backslash (Q \Rightarrow R) / \ni) \\
\mathit{step}\, S & = & \mathit{id} \;\cap\; \mathit{outl}^\circ \cdot S \cdot \mathit{outr}
\end{array}
\end{array}
$$

For this rule, not only a proof using points is more effective, but also the actual statement gets more readable. Application to the (pair of sets) point $\langle\!| A_1, A_2 |\!\rangle$ and some juggling transforms it to

$$
\left.
\begin{array}{r}
\mathit{mins} \cdot \Lambda(A_1 \cup A_2) \quad \supseteq \quad \mathit{cup} \cdot \langle \mathit{mins} \cdot \Lambda A_1, \mathit{mins} \cdot \Lambda(A_2 - Q \cdot A_1) \rangle \\
\text{provided} \quad A_1 \cdot A_2^\circ \cap Q \subseteq R
\end{array}
\right\} \tag{4.1}
$$

Let us now proceed with its proof. We will show that

$$(\forall\, b \;:\; b \subseteq \mathit{LHS} \;\Leftarrow\; b \subseteq \mathit{RHS})$$

The antecedent is first manipulated thus

$$
\begin{array}{cll}
 & b \subseteq \mathit{RHS} & \\
\equiv & \quad \{ \text{ points-composition } \} & \\
 & (\exists\, c \;:\; b \subseteq \mathit{cup} \cdot c \;\wedge\; c \subseteq \langle\, \mathit{mins} \cdot \Lambda A_1 \,,\, \mathit{mins} \cdot \Lambda(A_2 - Q \cdot A_1) \,\rangle\,) & \\
\equiv & \quad \{ \text{ naming convention, } \mathit{cup} \text{ property, products } \} & \\
 & (\exists\, C_1, C_2 \;:\; \Lambda B \subseteq \Lambda(C_1 \cup C_2) & \\
 & \qquad\qquad \wedge\; \Lambda C_1 \subseteq \mathit{mins} \cdot \Lambda A_1 \;\wedge\; \Lambda C_2 \subseteq \mathit{mins} \cdot \Lambda(A_2 - Q \cdot A_1)) & \\
\equiv & \quad \{ \text{ inclusion/equality of functions, power-transpose, pointwise } \mathit{mins} \text{ } \} & \\
 & (\exists\, C_1, C_2 \;:\; B = C_1 \cup C_2 & \text{(i)} \\
 & \qquad\qquad \wedge\; C_1 \cdot C_1^\circ \cap Q \subseteq \mathit{id} \;\wedge\; C_2 \cdot C_2^\circ \cap Q \subseteq \mathit{id} & \text{(ii)} \\
 & \qquad\qquad \wedge\; C_1 \subseteq A_1 \;\wedge\; A_1 \subseteq (Q \cap R)^\circ \cdot C_1 & \text{(iii)} \\
 & \qquad\qquad \wedge\; C_2 \subseteq A_2 - Q \cdot A_1 \;\wedge\; A_2 - Q \cdot A_1 \subseteq (Q \cap R)^\circ \cdot C_2) & \text{(iv)}
\end{array}
$$

Assuming it, we prove the consequent, which reads

$$B \cdot B^\circ \cap Q \subseteq \mathit{id} \;\wedge\; B \subseteq A_1 \cup A_2 \;\wedge\; A_1 \cup A_2 \subseteq (Q \cap R)^\circ \cdot B$$

We proceed with the first conjunct

$$
\begin{array}{ll}
 & B\cdot B^\circ \cap Q \\
= & \quad \{ \text{ (i), distribution } \} \\
 & (C_1\cdot C_1^\circ \cap Q) \cup (C_1\cdot C_2^\circ \cap Q) \cup (C_2\cdot C_1^\circ \cap Q) \cup (C_2\cdot C_2^\circ \cap Q) \\
\subseteq & \quad \{ \text{ (ii), claim: second and third operands equal } \emptyset \text{ } \} \\
 & id
\end{array}
$$

It suffices to prove either part of the claim since they are equivalent (take converses and note that Q is symmetric):

$$
\begin{array}{ll}
 & C_2\cdot C_1^\circ \cap Q \\
\subseteq & \quad \{ \text{ modular law } \} \\
 & (C_2 \cap Q\cdot C_1)\cdot C_1^\circ \\
\subseteq & \quad \{ \text{ first conjuncts of (iii) and (iv), substraction } \} \\
 & (\overline{Q\cdot A_1} \cap Q\cdot A_1)\cdot C_1^\circ \\
= & \quad \{ \text{ complements } \} \\
 & \emptyset
\end{array}
$$

For the second conjunct, we argue

$$
\begin{array}{ll}
 & B \\
\subseteq & \quad \{ \text{ (i), first conjuncts of (iii) and (iv) } \} \\
 & A_1 \cup (A_2 - Q\cdot A_1) \\
\subseteq & \quad \{ \text{ substraction } \} \\
 & A_1 \cup A_2
\end{array}
$$

The third and last conjunct is the key one that depends on the proviso of the rule:

$$
\begin{array}{ll}
 & A_1 \cup A_2 \\
= & \quad \{ \text{ complements } \} \\
 & A_1 \cup (A_2 \cap Q\cdot A_1) \cup (A_2 - Q\cdot A_1) \\
\subseteq & \quad \{ \text{ modular law } \} \\
 & A_1 \cup (A_2\cdot A_1^\circ \cap Q)\cdot A_1 \cup (A_2 - Q\cdot A_1) \\
\subseteq & \quad \{ \text{ the proviso is equivalent to } A_2\cdot A_1^\circ \cap Q \subseteq (Q\cap R)^\circ \text{ } \} \\
 & A_1 \cup (Q\cap R)^\circ\cdot A_1 \cup (A_2 - Q\cdot A_1) \\
\subseteq & \quad \{ \text{ second conjuncts of (iii) and (iv), transitivity of } (Q\cap R)^\circ \text{ } \} \\
 & (Q\cap R)^\circ\cdot C_1 \cup (Q\cap R)^\circ\cdot C_2 \\
= & \quad \{ \text{ distribution, (i) } \} \\
 & (Q\cap R)^\circ\cdot B
\end{array}
$$

We have completed the proof.

A corresponding proof in the point-free style turned out to be more than twice the length of the one above. The core of the matter, though, is that, in the same way that the statement of the rule is less readable without points, the proof was not very easy to work out. The one shown here is far more symbol-guided.

Mins/Uniq A set satisfying *uniq* has itself, and only itself, as a minimum-representatives subset. The point-free rule reads

$$mins \cdot uniq \;=\; uniq$$

and translates to

$$mins \cdot a \;=\; a \qquad \text{provided } \; uniq.a \tag{4.2}$$

We prove it as follows:

$$\begin{array}{ll}
 & b \subseteq mins \cdot a \\
\equiv & \quad \{ \text{ pointwise } mins \text{ } \} \\
 & B{\cdot}B^{\circ} \cap Q \subseteq id \;\;\wedge\;\; B \subseteq A \;\;\wedge\;\; A \subseteq (Q \cap R)^{\circ}{\cdot}B \qquad (*) \\
\equiv & \quad \{ \text{ claim } \} \\
 & B = A \\
\equiv & \quad \{ \text{ power-transpose, inclusion/equality of functions } \} \\
 & b \subseteq a
\end{array}$$

The ($\Leftarrow$) part of the claim follows directly from the proviso and reflexivity of $(Q \cap R)^{\circ}$. For ($\Rightarrow$) we only need to prove $A \subseteq B$:

$$\begin{array}{ll}
 & A \\
= & \quad \{ \text{ third conjunct of } (*) \text{, meet } \} \\
 & A \;\cap\; (Q \cap R)^{\circ}{\cdot}B \\
\subseteq & \quad \{ \text{ meet, } Q \text{ symmetric } \} \\
 & A \;\cap\; Q{\cdot}B \\
\subseteq & \quad \{ \text{ modular law } \} \\
 & (A{\cdot}B^{\circ} \cap Q){\cdot}B \\
\subseteq & \quad \{ \text{ second conjunct of } (*) \text{ } \} \\
 & (A{\cdot}A^{\circ} \cap Q){\cdot}B \\
\subseteq & \quad \{ \text{ proviso } \} \\
 & B
\end{array}$$

Note that we have proved $thin \cdot uniq = uniq$ as well, since the first conjunct of (*) was never used.

Mins/Empty and Mins/Unitary Particular instances of sets satisfying *uniq* are the empty set and any singleton set. Those facts will be useful in the next chapter. For $empty = \Lambda\emptyset$ and $\tau = \Lambda id$, we have

$$mins \cdot empty \;=\; empty \tag{4.3}$$

$$mins \cdot \tau \;=\; \tau \tag{4.4}$$

4.3 Discussion

I wonder whether the assumption of atomicity is really necessary in the presence of tabulations. The only purpose it served was to make the point-inclusion and point-composition laws valid. I doubt atomicity follows from the existence of tabulations, but it would suffice that it implies the other above-mentioned laws.

There is yet another aspect of points: definition of relations through their pointwise extension. Given an indeterminate x typed $\mathrm{F{\scriptsize UN}}(A, 1)$ and an expression $\theta(x)$ in $\mathrm{R{\scriptsize EL}}(B, 1)$, there exists a unique relation $R : B \leftarrow A$ such that $R \cdot x = \theta(x)$ for all x. This allows us to define relations by means of their application to points, not having to give a closed point-free expression for them.

5 Main Derivation

We now present the core of the calculation of algorithmic solutions to the minimum-representatives of closure problem:

$$P \;=\; mins(Q,R)\cdot \Lambda S^*$$

In order to successfully promote the *mins* selector into the generator, the closure operation is first suitably generalised. Following that, a key property that relates the thinning part of *mins* to the generalised closure is stated. All the properties gathered to this point are then put to use deriving a recursion equation satisfied by P (or, rather, a generalisation of it).

5.1 Generalising Closure

Recall that, by definition, S^* is the least preorder that contains S. And that it is the least solution of the equation $S^* = (\mu X : \mathit{id} \cup S{\cdot}X)$, which is a particular case of the more general result

$$S^*{\cdot}A \;=\; (\mu X : A \,\cup\, S{\cdot}X) \tag{5.1}$$

If A is a powerset point, i.e. a set, $S^*{\cdot}A$ can be thought of as the right-to-left image through S^* of A. For the applications we have in mind, algorithms that compute P, the above generalisation will not lead to a terminating recursion. Termination will be ensured by finiteness of the set of equivalence classes of Q. Hence, we should somehow take account of Q-classes that "need not be generated anymore". Accordingly, we choose to generalise (5.1) as follows

$$\Theta(B,A) \;\widehat{=}\; (\mu X : A \,\cup\, (S{\cdot}X - Q{\cdot}B))$$

The rest of the section is devoted to state and prove properties of Θ which will be put to use later.

We start out with equations that will allow us to embed P into a suitable generalisation and afterwards carry out a recursion analysis:

$$\Theta(\emptyset,A) \;=\; S^*{\cdot}A \tag{5.2}$$

$$\Theta(B,\emptyset) \;=\; \emptyset \tag{5.3}$$

$$\Theta(B,A_1\cup A_2) \;=\; A_1 \,\cup\, \Theta(B,A_2 \,\cup\, (S{\cdot}A_1 - Q{\cdot}B)) \tag{5.4}$$

(5.2) follows from $\emptyset$ being a zero of composition and right-unit of substraction, and equation (5.1). Since $\emptyset$ is also the unit of join and a left-zero of substraction, note for (5.3) that $\emptyset$ is a solution of the recursion equation involved and therefore the least solution. Finally, the rolling rule of the μ-calculus

$$(\mu X : \phi(\psi X)) \;=\; \phi(\mu X : \psi(\phi X))$$

is used to prove (5.4):

$$\begin{array}{cl} & \Theta(B, A_1 \cup A_2) \\ = & \quad \{ \text{ definition of } \Theta \ \} \\ & (\mu X : A_1 \ \cup \ A_2 \ \cup \ (S{\cdot}X - Q{\cdot}B)) \\ = & \quad \{ \text{ rolling with } \ \phi X := A_1 \cup X, \ \ \psi X := A_2 \cup (S{\cdot}X - Q{\cdot}B) \ \} \\ & A_1 \ \cup \ (\mu X : A_2 \ \cup \ (S{\cdot}(A_1 \cup X) - Q{\cdot}B)) \\ = & \quad \{ \text{ distribution } \} \\ & A_1 \ \cup \ (\mu X : A_2 \ \cup \ (S{\cdot}A_1 - Q{\cdot}B) \ \cup \ (S{\cdot}X - Q{\cdot}B)) \\ = & \quad \{ \text{ definition of } \Theta \ \} \\ & A_1 \ \cup \ \Theta(B, A_2 \ \cup \ (S{\cdot}A_1 - Q{\cdot}B)) \end{array}$$

We shall also need the following property:

$$\Theta(B, A) \ \subseteq \ S^{*}{\cdot}A \tag{5.5}$$

which follows from $X - Y \subseteq X$, monotonicity of the μ-operator and (5.1). To close this list of properties, we point out that Θ is anti-monotonic on its first argument and monotonic on its second.

Thinning the Closure

A crucial property of Θ in connection to the thinning part of the *mins* selector follows. Such property relies on the parameters Q, R, S of our problem satisfying two conditions:

$$S \ \subseteq \ R^{\circ} \tag{Hyp.1}$$

$$S \cdot (Q \cap R)^{\circ} \ \subseteq \ (Q \cap R)^{\circ} \cdot S \tag{Hyp.2}$$

The first equation says that the generating relation S does not decrease the R-cost. Therefore, minimum-representatives are likely to be produced sooner rather than later. The second equation states the monotonicity of S on the converse of the thinning mediator $(Q \cap R)^{\circ}$. This permits the disposal of some intermediate candidates for the presence of others. If one has at some stage x and y such that $x \, (Q \cap R)^{\circ} \, y$, i.e., they are equivalent and y is at least as good as x, then for any element x' generated from x there will be a y' generated by y equivalent to and at least as good as x'. Therefore, x can be safely disposed of. These two assumptions allow the (closure-)generating process to be pruned considerably.

The *thin*/Θ rule formalises the pruning we shall make use of. It reads

$$\left.\begin{array}{r} \mathit{thin}\,(Q \cap R) \ \cdot \ \Lambda\,(\Theta(B, A) - Q{\cdot}C) \ \ \supseteq \ \ \Lambda\,\Theta(B \cup C, A - Q{\cdot}C) \\ \text{provided} \ \ C{\cdot}A^{\circ} \subseteq R \quad\quad \\ S{\cdot}C - Q{\cdot}B \subseteq A \quad\quad \end{array}\right\} \tag{5.6}$$

Its proof can be found in appendix A.

Note that, from (Hyp.1), we have in R° a preorder that contains S. Therefore, by definition of closure, $S^* \subseteq R^\circ$ and (5.5) then gives $\Theta(B, A) \subseteq R^\circ \cdot A$. It follows that, by transitivity of R,

$$C \cdot A^\circ \subseteq R \quad \Rightarrow \quad C \cdot \Theta(B, A)^\circ \subseteq R \tag{5.7}$$

Finally, we shall also use the fact that *thin* is monotonic in the sense that $A \subseteq B$ implies $thin \cdot \Lambda A \subseteq thin \cdot \Lambda B$.

5.2 Main Derivation

We now manipulate the whole of our abstract problem P, first embedding it into an adequate generalisation and then deriving a recursion equation for the latter. Recall the hypotheses (Hyp.1) and (Hyp.2), which act as provisos for the derivation to come.

The calculation makes use of pointwise reasoning. We take a point a and proceed thus with the embedding

$$\begin{array}{ll} & P \cdot a \\ = & \quad \{ \text{ definition of } P \text{, power-transpose absorption } \} \\ & mins \cdot \Lambda(S^* \cdot a) \\ = & \quad \{ \; \Theta \text{ embeds closure (5.2) } \} \\ & mins \cdot \Lambda\, \Theta(\emptyset, a) \\ = & \quad \{ \text{ introduce } P'(B, A) := mins \cdot \Lambda\, \Theta(B, A) \; \} \\ & P'(\emptyset, a) \end{array}$$

In the third step two things were actually introduced: a relation P' satisfying $P' \cdot \langle\!| B, A |\!\rangle = mins \cdot \Lambda\, \Theta(B, A)$ and notation $P'(B, A) := P' \cdot \langle\!| B, A |\!\rangle$. A case analysis (on A) is now conducted for P'. First, the empty set case. By Θ property (5.3) and *mins*/*empty* rule (4.3), we have

$$P'(B, \emptyset) \;=\; mins \cdot \Lambda\, \Theta(B, \emptyset) \;=\; mins \cdot \Lambda\emptyset \;=\; \Lambda\emptyset$$

Second, the case of A being the union of two sets. This case includes the first one, so further refinement in later steps will be needed to eliminate the non-determinism. For the time being, it is kept simple. We use a heuristic analoguous to the one used for the *mins*/*cup* rule. From the input set ($A = A_1 \cup A_2$) a part is processed first (A_1), taking representatives of it, and then carrying on with the rest. Conditions under which it is valid to do so will be calculated. So, let us take A_1' such that $\Lambda A_1' \subseteq mins \cdot \Lambda A_1$ and then proceed:

$$
\begin{array}{cl}
& P'(B, A_1 \cup A_2) \\
= & \quad \{ \text{ definition of } P' \ \} \\
& \mathit{mins} \cdot \Lambda\, \Theta(B, A_1 \cup A_2) \\
\supseteq & \quad \{ \ A_1 \supseteq A_1' \text{ by assumption, monotonicity of } \Theta \text{ and } \mathit{thin} \ \} \\
& \mathit{mins} \cdot \Lambda\, \Theta(B, A_1' \cup A_2) \\
= & \quad \{ \ \Theta \text{ property (5.4) } \} \\
& \mathit{mins} \cdot \Lambda\,(\, A_1' \,\cup\, \Theta(B, A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B))\,) \\
\supseteq & \quad \left\{ \begin{array}{l} \mathit{mins}/\mathit{cup} \text{ rule (4.1)} \\ \bullet \text{ assumption: proviso of the rule} \end{array} \right\} \\
& \mathit{cup} \cdot \langle\, \mathit{mins} \cdot \Lambda A_1' \,,\ \mathit{mins} \cdot \Lambda\,(\, \Theta(B, A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B)) \,-\, Q{\cdot}A_1' \,)\,\rangle \\
\supseteq & \quad \left\{ \begin{array}{l} \mathit{mins}/\mathit{uniq} \text{ rule (4.2), } A_1' \text{ satisfies } \mathit{uniq} \text{ by assumption;} \\ \mathit{thin}/\Theta \text{ rule (5.6), } \ \mathit{thin}{\cdot}X \supseteq Y \;\Rightarrow\; \mathit{mins}{\cdot}X \supseteq \mathit{mins}{\cdot}Y \\ \bullet \text{ assumption: provisos of the rule} \end{array} \right\} \\
& \mathit{cup} \cdot \langle\, \Lambda A_1' \,,\ \mathit{mins} \cdot \Lambda\, \Theta(B \cup A_1',\, (A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B)) \,-\, Q{\cdot}A_1')\,\rangle \\
= & \quad \{ \text{ definition of } P'\text{, distribution } \} \\
& \mathit{cup} \cdot \langle\, \Lambda A_1' \,,\ P'(B \cup A_1',\, (A_2 - Q \cdot A_1') \,\cup\, (S{\cdot}A_1' - Q{\cdot}B - Q \cdot A_1'))\,\rangle
\end{array}
$$

The additional assumptions made are now collected and analysed. These are

$$
\begin{array}{llr}
A_1' \cdot \Theta(B, A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B))^\circ \,\cap\, Q \;\subseteq\; R & & \text{(i)} \\
A_1' \cdot (A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B))^\circ \;\subseteq\; R & & \text{(ii)} \\
S{\cdot}A_1' - Q{\cdot}B \;\subseteq\; A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B) & & \text{(iii)}
\end{array}
$$

First note that (i) follows from (ii), by an appeal to (5.7) and universal property of meet. And (iii) is true for universal property of join. So we are left only with (ii), but it can be refined to a condition both simpler and only in terms of the actual input (i.e. not involving A_1'):

$$
\begin{array}{cll}
& A_1' \cdot (A_2 \,\cup\, (S{\cdot}A_1' - Q{\cdot}B))^\circ \;\subseteq\; R & \\
\Leftarrow & \quad \{ \text{ substraction } \} & \\
& A_1' \cdot (A_2 \,\cup\, S{\cdot}A_1')^\circ \;\subseteq\; R & \\
\Leftarrow & \quad \{ \ R \text{ reflexive, (Hyp.1) } \} & \\
& A_1' \cdot (R^\circ{\cdot}A_2 \,\cup\, R^\circ{\cdot}A_1')^\circ \;\subseteq\; R & \\
\Leftarrow & \quad \{ \text{ distribution, converse } \} & \\
& A_1' \cdot (A_2 \cup A_1')^\circ \cdot R \;\subseteq\; R & \\
\Leftarrow & \quad \{ \ R \text{ transitive, monotonicity } \} & \\
& A_1' \cdot (A_2 \cup A_1')^\circ \;\subseteq\; R & \\
\Leftarrow & \quad \{ \ A_1' \subseteq A_1 \text{ by assumption } \} & \\
& A_1 \cdot (A_1 \cup A_2)^\circ \;\subseteq\; R & \text{(ii')}
\end{array}
$$

In order to state the recursion (in)equation derived above, we now move to the point-free space. The results translate to

$$
\begin{array}{rcl}
P' \cdot (id \times empty) & = & empty! \\
P' \cdot (id \times cup) & \supseteq & cup \cdot (id \times P') \cdot combine \cdot (id \times (mins \times id) \cdot p?)
\end{array}
$$

where *combine* and $p?$ are, respectively, the function and coreflexive defined as

$$
\begin{array}{rcl}
combine \cdot \langle\!| B, A_1, A_2 |\!\rangle & = & \langle\!| A_1, B \cup A_1, (A_2 - Q{\cdot}A_1) \cup (S{\cdot}A_1 - Q{\cdot}B - Q{\cdot}A_1) |\!\rangle \\
p?.\, \langle\!| A_1, A_2 |\!\rangle & \equiv & \text{(ii')}
\end{array}
$$

Now note that

$$
\begin{array}{cl}
 & P' \\
\supseteq & \quad \{ \text{ above, shunting of functions, join } \} \\
 & empty! \cdot (id \times empty^{\circ}) \ \cup \\
 & cup \cdot (id \times P') \cdot combine \cdot (id \times (mins \times id) \cdot p? \cdot cup^{\circ}) \\
= & \quad \{ \text{ coproducts } \} \\
 & [\, empty!\, ,\ cup\,] \cdot (id + id \times P') \cdot \\
 & [\, (id \times empty)\, ,\ (combine \cdot (id \times (mins \times id) \cdot p? \cdot cup^{\circ}))^{\circ}]^{\circ} \\
= & \quad \{ \text{ introducing suitable function } g\text{, functor } \mathsf{F} \text{ and relation } W \ \} \\
 & g \cdot \mathsf{F}P' \cdot W
\end{array}
$$

and that

$$
\begin{array}{cl}
 & P \\
= & \quad \{ \text{ point-free version of the embedding } \} \\
 & P' \cdot \langle\, empty!\, ,\ id \,\rangle \\
= & \quad \{ \text{ introduce function } h \ \} \\
 & P' \cdot h \\
\supseteq & \quad \{ \text{ above inequation for } P'\text{, Knaster-Tarski } \} \\
 & (\mu X : g \cdot \mathsf{F}X \cdot W) \cdot h
\end{array}
$$

The core of the calculation has been completed.

5.3 Towards Functional Refinements

In the relational-algebraic approach to derivation of algorithms one starts with a relational, and so potentially non-deterministic, specification P and aims to refine it to a (possibly partial) function, thus deterministic, p. The refinement must respect the original specification, which is formalised through inclusion, and should also work for the same set of inputs. A domain-preserving refinement relation can be defined as

$$T \succeq T' \quad \widehat{=} \quad T \supseteq T' \ \wedge \ dom\,T = dom\,T'.$$

The domain of a relation R is the coreflexive $dom\,R = id \cap R^{\circ} \cdot R$ and, dually, its range is $ran\,R = dom\,R^{\circ}$. What we want for a specification P are implementations[3] p of it in the sense that

$$P \text{ is implemented by } p \quad \widehat{=} \quad P \succeq p \ \wedge \ p \text{ is simple.}$$

For the representatives of closure problem P we have already derived that $P \supseteq (\mu X : g \cdot \mathsf{F}X \cdot W) \cdot h$. Equality does not hold in general, so some no-determinism has been swept away. Yet not all of it since W is not a simple relation. If we are to hope this is leading us to an implementation, we need to check the domain has been preserved so far. For the subsequent domains analysis we impose a new restriction on the mediator parameter R. It is assumed to be well-founded:

$$dom \in \quad \subseteq \quad dom\,(min\,R) \tag{Hyp.3}$$

Under this new assumption P is entire and so we need entire implementations, i.e. (total) functions. To analyse the domain-preservation, domains of recursion equations need be reasoned about. Inductiveness is the tool we shall use for that purpose.

A relation T is inductive if there are no infinite chains $a_0, a_1, a_2, \ldots$ such that $(\forall i : i \geq 0 : a_{i+1} T a_i)$.[4] For a monotonic functor F, T is said to be F-inductive if $\delta \cdot T$ is inductive, where δ is the membership relation of F (see [23]). Inductiveness is conventionally used to prove termination of programs. In a relational framework, termination is expressed through domains of recursion equations, which we formalise thus:

Proposition 5.8 *Let R, S be relations and C a coreflexive such that R is entire, $C \subseteq dom(\mathsf{F}C \cdot S)$ and S is F-inductive. Then $C \subseteq dom(\mu X : R \cdot \mathsf{F}X \cdot S)$.*

The proof of this result is in appendix B.

We now have an informal look at the F-inductiveness of W. We do so taking an input point to the recursion and finding out what the input is for the next

[3] This is a very loose notion of implementation, since it does not refer to any computational mechanisms or notions of what is executable. It must be understood in such a general sense.

[4] This is actually the (informal) characterisation of well-founded relations. However, inductiveness and well-foundedness are equivalent in the presence of negation. See [3, 21] for a very good calculational presentation of these concepts. We reserve the term well-founded for the (related) concept expressed in (Hyp.3).

recursive level, seeing what is "below" such point under $\delta \cdot W$:

$$
\begin{array}{cl}
& \delta \cdot W \cdot \langle\!| B, A |\!\rangle \\
= & \left\{ \begin{array}{l} \text{membership relations: } \delta_{\mathsf{F}} = [\emptyset, outr], \\ \text{definition of } W \text{, coproducts} \end{array} \right\} \\
& outr \cdot combine \cdot (id \times (mins \times id) \cdot p? \cdot cup^\circ) \cdot \langle\!| B, A |\!\rangle \\
\supseteq & \left\{ \begin{array}{l} \text{let } A_1, A_2 \text{ be such that} \\ \qquad - \;\; \langle\!| A_1, A_2 |\!\rangle \subseteq cup^\circ \cdot \langle\!| A |\!\rangle \;\; (\equiv \; (A_1 \cup A_2 = A)) \\ \qquad - \;\; p?.\langle\!| A_1, A_2 |\!\rangle \\ \text{let } A'_1 \text{ be such that} \\ \qquad - \;\; \langle\!| A'_1 |\!\rangle \subseteq mins \cdot \langle\!| A_1 |\!\rangle \end{array} \right\} \\
& outr \cdot combine \cdot \langle\!| B, A'_1, A_2 |\!\rangle \\
= & \{ \text{ definition of } combine \text{, products } \} \\
& \langle\!| B \cup A'_1, (A_2 - Q \cdot A'_1) \cup (S \cdot A'_1 - Q \cdot B - Q \cdot A'_1) |\!\rangle
\end{array}
$$

Hence, $\langle\!| B \cup A'_1, \ \ldots |\!\rangle \subseteq \delta \cdot W \cdot \langle\!| B, A |\!\rangle$ provided A'_1 can be extracted from A under the above restrictions. We focus on the first component of the pair and note it seems likely to grow. However, we have to show that it does grow and that such progress is bounded.

First of all, we need A'_1 to be non-empty. This is solved refining W to W' by restricting $p?$ to $p? \cdot (dom \in \times id)$. Now, if we knew A'_1 is not contained in B the growing would be guaranteed. But this would not ensure termination yet since such growing is not taking place within a finite set. (In the instance of minimum paths, the set of paths is, in general, infinite.) However, recall the purpose of that first component is to record the Q-equivalence classes representatives of which have already been sent to the output. Therefore, one would expect A'_1 to be recording new outputted classes, that is, not to share classes with B. This is indeed the case, since invariantly A (from which A'_1 was extracted) and B do not share Q-classes. Define the Q-disjoint coreflexive as

$$disj\, Q \quad \widehat{=} \quad step\, (\in \backslash \overline{Q} / \ni),$$

which pointwisely reads

$$
\begin{array}{rcl}
disj\, Q.\langle\!| B, A |\!\rangle & \equiv & B \cdot A^\circ \subseteq \overline{Q} \\
& \equiv & B \cdot A^\circ \cap Q = \emptyset.
\end{array}
$$

We claim that *disj* Q is an invariant of the refined algorithm. A brief review of the concept of invariants in this framework follows.

Invariants

Invariants are important in the development of imperative programs, specifically, in the development of loops. An invariant is often just a generalisation of the postcondition to be achieved, that must be set before entering the loop-to-be and maintained after each iteration of the loop-body-to-be. The generalisation must be carefully chosen in order to serve such purpose. In most of the work done in relational derivation of functional algorithms, the concept of invariant seems to disappear. In our specific development of the representatives problem, the success of the calculation was due to the choice of generalisation of the closure operator (definition of Θ in section 5.1). This generalisation of (part of) the specification to get the derivation rolling can be seen as the parallel of the imperative choice of a postcondition-generalised invariant to develop a loop.

Nevertheless, invariants turned out to be needed towards the end of our development: to prove domain-preservation. If a entire specification is refined to a least fixed point, the latter is expected to be entire. A practical result previously used within BdM's calculus to show certain refinements preserve domains is the following: $(\mu X : R \cdot \mathsf{F}X \cdot S)$ is entire if R and S are entire and S is F-inductive. In our representatives example, we hit a similar recursion equation where S is not F-inductive unless restricted to certain inputs. But the refinement is of the form $(\mu X : R \cdot \mathsf{F}X \cdot S) \cdot T$, and T delivers only the kind of outputs that guarantee S to terminate. I.e., T delivers results within a certain coreflexive C, S will guarantee that C keeps holding when entering the next recursion level, and the recursion will terminate due to $S \cdot C$ being F-inductive. Comparing again to the imperative approach, we are in the presence of an invariant the purpose of which is showing the variant decreases, rather than showing the output of the computation is the desired one. In Bird's imperative presentation of the representatives problem [9], both kinds of invariants are used.

Formally, a coreflexive C is said to be F-invariant under S if $S \cdot C \subseteq \mathsf{F}C \cdot S$. Also, C is said to be set by T if $ran\, T \subseteq C$ (equivalent to $T = C \cdot T$). We now proceed with two small results.

Proposition 5.9 *If C is F-invariant under S, then*

$$(\mu X : R \cdot \mathsf{F}X \cdot S) \cdot C \quad = \quad (\mu X : R \cdot \mathsf{F}X \cdot S \cdot C)$$

The proof is in appendix B. An immediate corollary:

Proposition 5.10 *If C is both F-invariant under S and set by T, then*

$$(\mu X : R \cdot \mathsf{F}X \cdot S) \cdot T \quad = \quad (\mu X : R \cdot \mathsf{F}X \cdot S \cdot C) \cdot T$$

Now, the right proposition to prove our refinements are domain-preserving.

Proposition 5.11 *Let R, S, T be entire relations and C be a coreflexive. Suppose C is both F-invariant under S and set by T, and such that $S \cdot C$ is F-inductive. Then $(\mu X : R \cdot \mathsf{F}X \cdot S) \cdot T$ is entire.*

We argue

$$
\begin{array}{ll}
 & (\mu X : R \cdot \mathsf{F}X \cdot S) \cdot T \ \text{ entire} \\
\equiv & \quad \{ \text{ proposition (5.10) } \} \\
 & (\mu X : R \cdot \mathsf{F}X \cdot S \cdot C) \cdot T \ \text{ entire} \\
\Leftarrow & \quad \{ \ T \text{ entire, domains } \} \\
 & \mathit{ran}\, T \ \subseteq \ \mathit{dom}\, (\mu X : R \cdot \mathsf{F}X \cdot S \cdot C) \\
\Leftarrow & \quad \{ \ C \text{ is set by } T \ \} \\
 & C \ \subseteq \ \mathit{dom}\, (\mu X : R \cdot \mathsf{F}X \cdot S \cdot C) \\
\Leftarrow & \quad \{ \text{ proposition (5.8), } R \text{ entire, } S \cdot C \ \ \mathsf{F}\text{-inductive } \} \\
 & C \ \subseteq \ \mathit{dom}\, (\mathsf{F}C \cdot S \cdot C) \\
\Leftarrow & \quad \{ \ C \ \mathsf{F}\text{-invariant under } S \text{, duplicity of coreflexives } \} \\
 & C \ \subseteq \ \mathit{dom}\, (S \cdot C) \\
\equiv & \quad \{ \ S \text{ entire } \} \\
 & \mathit{true}
\end{array}
$$

A last property of invariants: they are inherited through inclusion. That is, if $S' \subseteq S$ and C is F-invariant under S, then C is also F-invariant under S'.

Getting back again to the representatives algorithm, from the main derivation and the last refinement to W, it follows that

$$P \quad \supseteq \quad (\mu X : g \cdot \mathsf{F}X \cdot W') \cdot h$$

We now claim that (i) $\mathit{disj}\, Q$ is F-invariant under W' and set by h, (ii) W' is entire, and (iii) $W' \cdot \mathit{disj}\, Q$ is F-inductive. Since g and h are functions, thus entire, it follows from proposition (5.11) that

$$P \quad \succeq \quad (\mu X : g \cdot \mathsf{F}X \cdot W') \cdot h$$

The proof of (i) is omitted. For (ii), a close look at W' shows that the empty set gets injected to the left and all non-empty sets to the right. The feasibility of dividing sets in two parts satisfying $p? \cdot (\mathit{dom} \in \times \mathit{id})$ is guaranteed by (Hyp.3). The proof of (iii) is also omitted, but we point out that requires yet another condition:

$$\text{The quotient set of } Q \text{ is finite} \qquad \text{(Hyp.4)}$$

5.4 Two Particular Implementations

We aim to get two particular kinds of implementation for our abstract problem, which we name *unitary* and *levelled*. Note that all the non-determinism left is in the subexpression $(mins \times id) \cdot p? \cdot (dom \in \times\, id) \cdot cup^\circ$ of W'. It says: split (cup°) the input set, extracting a non-empty ($dom \in$) subset the elements of which are R-minima ($p?$), take representatives ($mins$) of those minima and then carry on. Roughly, the unitary and levelled implementations will respectively do so thus:

$$\begin{array}{rcl} \{a\} \cup x & \leadsto & \{a\}\,, x \\ x & \leadsto & x\,, \emptyset \end{array}$$

Recall the class of graph problems we abstracted P from in chapter 3. The unitary solution provides Dijkstra's minimum paths algorithm and both depth-first and breadth-first reachability algorithms. From the levelled solution, the breadth-first search and unweighted shortest paths algorithms that work by levels are obtained. Conditions that suffice to get an implementation, i.e. to adequately refine W', follow:

Let $split$ be a simple relation and $q?$ be a coreflexive such that

$$split \cdot q? \;\subseteq\; (mins \times id) \cdot p? \cdot (dom \in \times\, id) \cdot cup^\circ \tag{i}$$

$$dom\, split \;=\; dom \in \tag{ii}$$

$$combine \cdot split \cdot q? \;\subseteq\; (id \times q?) \cdot combine \cdot split \tag{iii}$$

$$h \;\subseteq\; q? \cdot h \tag{iv}$$

Then

P is implemented by $M \cdot h$
where

$$\begin{array}{rcl} M & = & (\mu X : g \cdot \mathsf{F}X \cdot V) \\ V & = & [\,(id \times empty)\,, (combine \cdot split)^\circ\,]^\circ \end{array}$$

Three things need be shown: $P \supseteq M \cdot h$, $dom\, P = dom\,(M \cdot h)$, and that $M \cdot h$ is simple.

First note that (i) and (iii) imply, respectively, (i') $V \cdot q? \subseteq W'$ and (iii') $V \cdot q? \subseteq \mathsf{F}q? \cdot V$. Then

$$\begin{array}{cl} & M \cdot h \\ = & \quad \{ \text{ (iii'), (iv), proposition (5.10) } \} \\ & (\mu X : g \cdot \mathsf{F}X \cdot V \cdot q?) \cdot h \\ \subseteq & \quad \{ \text{ (i') } \} \\ & (\mu X : g \cdot \mathsf{F}X \cdot W') \cdot h \\ \subseteq & \quad \{ \text{ last section } \} \\ & P \end{array}$$

Next, note that, by (i'), $V \cdot q?$ inherits from W' its two properties in relation to $disj\; Q$: having it as F-invariant and being F-inductive when restricted to it. Therefore, $(q? \cdot disj\; Q)$ is F-invariant under V and $V \cdot (q? \cdot disj\; Q)$ is F-inductive.

And V is entire by (ii). We have got the provisos of proposition (5.11) that show $M \cdot h$ to be entire. Finally, $M \cdot h$ simple follows from *split* being simple.

Now for the two solutions...

Unitary Split Let *getmin* be a simple relation such that

$$getmin \quad \subseteq \quad step\,(\tau \cdot min\, R) \cdot cup^{\circ}$$

Then $split, q? := getmin, id$ provide an implementation as required above.

Full Split (Levelled Solution) Let r be a function such that $r \subseteq mins\,(Q, \Pi)$ and define *getlevel* to be

$$getlevel \quad = \quad \langle\, r, empty!\, \rangle \cdot dom \in$$

Then $split, q? := getlevel, unif\ R$ where $unif\ R$ is the coreflexive denoting sets of elements with uniform R-cost, satisfy the provisos to get an implementation. For $unif\ R$ to be invariant an extra condition must be satisfied:

$$S \cdot R \cdot S^{\circ} \quad \subseteq \quad R \tag{Hyp.Extra}$$

To implement the derived algorithms in a functional language like Orwell or Gofer, the sets involved would have to be data-refined to, for instance, lists. The generating relation S should be required to spawn finite sets in order to fulfill this last step of the development:

$$ran\ \Lambda S \quad \subseteq \quad finite? \tag{Hyp.5}$$

All the instance problems of chapter 3 satisfy the five conditions required for the unitary implementation. The special requirement (Hyp.Extra) for a levelled solution is met by all but the first –minimum paths over a weighted graph–. For the rooted trees problem, it should be proved that the derived algorithms establish the additional post-condition needed.

6 Conclusions

We have specified the abstract problem of finding minimum-representatives of closure, and identified a few graph problems as instances of it. Also, algorithms to compute such representatives have been derived, requiring some conditions on the parameters of the specification. The task of obtaining algorithmic solutions for instance problems is reduced to verify the provisos are met by the particular parameters. This approach to algorithm design compares positively to other conventional approaches (see e.g. [16, 29, 47, 48]) both for its algebraic nature and the higher level of abstraction used. Algebraic presentations of algorithm design facilitates the visualisation of the points where crucial design decision are made. The higher level of abstraction permits the solution of several, sometimes seemingly disparate, problems all at once. The relational calculus of Bird and de Moor provided an adequate algebraic framework to carry out the development. However, its point-free nature sometimes limited the clarity of expression. Categorical points provided a smooth way of extending the calculus to ease such difficulty. Also, a few propositions regarding the use of invariants to reason about termination and domain-preservation of refinements were stated and proved.

The graph problems presented here are clearly related to each other. Some conventional presentations of graph algorithmics actually call the attention to the relationship between them; however, this is often done at the algorithm level rather than at the specification level, which we believe is the right place to do so. In chapter 3 the specification of each problem was developed separately, without showing much of the conceptual link between them. The missing point is that all can be regarded as path problems. The following equations hold for the generators given ($gen\, i$ denotes the generator of problem $3.i$):

$$\begin{aligned} gen2 &= (\![\, wrap\,,\, cons \cdot (to \times id)\,]\!) \cdot gen1 \\ gen3 &= (\![\, id\,,\, to \cdot outl\,]\!) \cdot gen1 \\ gen4 &= (\![\, newe\,,\, orige \cdot outl\,]\!) \cdot gen1 \end{aligned}$$

This is much in the same spirit of [42], where the original abstract specification is already bound to paths. Since the representatives of closure problem in itself does not deal with paths, its applicability to problems of other kinds is not restricted beforehand. Oege de Moor has suggested its application to a bus-stops problems drawn from the operations research field.

Others have studied this same class of graph problems. Backhouse et al have done so using regular algebra of matrices, by means of a correspondence between graphs and adjacency matrices [5]. Möller and Russling have tackled it using another relational framework [38, 42]; however, the weighted graph case is not considered and only breadth-first solutions are shown. Clenaghan showed that the two aforementioned approaches can be formally unified using a dynamic algebra framework [15].

To what extent graph algorithms can be classified, according to the design strategies that govern their construction, is an important goal to pursue. The present work encourages the undertaking of more research in such direction using the relational calculus. For instance, the connection to depth-first search can be used to explore the variety of algorithms based on it. King and Launchbury, in [28], give functional implementations of some DFS-based algorithms. They show how the change of presentation ease the task of reasoning about such programs, compared to traditional imperative views. However, no derivation of the algorithms are presented.

References

[1] C.J. Aarts. Galois connections presented calculationally. Graduating Dissertation, Eindhoven University of Technology, July 1992.

[2] C.J. Aarts, R.C. Backhouse, P. Hoogendijk, E. Voermans, and J. van der Woude. A relational theory of datatypes. Working document, December 1992. Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`.

[3] R.C. Backhouse and H. Doornbos. Mathematical induction made calculational. CS report 94-16, Eindhoven University of Technology, March 1994. Submitted for publication.

[4] R.C. Backhouse, H. Doornbos, and P. Hoogendijk. A class of commuting relators. Department of Mathematics and Computing Science, Eindhoven University of Technology, September 1992. Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`.

[5] R.C. Backhouse, J.P.H.W. van den Eijnde, and A.J.M. van Gasteren. Calculating path algorithms. *Science of Computer Programming*, 22(1-2), April 1994.

[6] J. Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8), August 1978.

[7] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice-Hall, 2nd edition, 1995.

[8] R.S. Bird. A calculus of functions for program derivation. In D.A. Turner, editor, *Research Topics in Functional Programming*. Addison-Wesley, 1990.

[9] R.S. Bird. On computing representatives. Oxford University Computing Laboratory, Programming Research Group, 1995.

[10] R.S. Bird and O. de Moor. Between dynamic programming and greedy: Data compression. Oxford University Computing Laboratory, Programming Research Group, 1992.

[11] R.S. Bird and O. de Moor. From dynamic programming to greedy algorithms. In B. Möller, H. Partsch, and S. Schumann, editors, *State-of-the-Art Report on Formal Program Development*. Springer-Verlag LNCS 755, 1992.

[12] R.S. Bird and O. de Moor. Solving optimisation problems with catamorphisms. In R.S. Bird, C.C. Morgan, and J.C.P. Woodcock, editors, *Mathematics of Program Construction*. Springer-Verlag LNCS 669, 1992.

[13] R.S. Bird and O. de Moor. *The Algebra of Programming.* Prentice-Hall, to appear 1996.

[14] R.S. Bird, P. Hoogendijk, and O. de Moor. Generic programming with relations and functors. Submitted to Journal of Functional Programming, 1993.

[15] K. Clenaghan. Calculational graph algorithmics: Reconciling two approaches with dynamic algebra. Report CS-R9518, CWI, March 1995.

[16] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[17] S. Curtis and G. Lowe. A graphical calculus. In B. Möller, editor, *Mathematics of Program Construction.* Springer-Verlag LNCS 947, 1995.

[18] O. de Moor. *Categories, Relations and Dynamic Programming.* PhD thesis, Oxford University Computing Laboratory, Programming Research Group, 1992.

[19] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1959.

[20] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[21] H. Doornbos and R.C. Backhouse. Induction and recursion on datatypes. In B. Möller, editor, *Mathematics of Program Construction.* Springer-Verlag LNCS 947, 1995.

[22] L.R. Foulds. *Graph Theory Applications.* Springer-Verlag, 1992.

[23] P. Freyd, O. de Moor, and P. Hoogendijk. Membership of datatypes. Unpublished Draft.

[24] P. Freyd and A. Ščedrov. *Categories, Allegories.* North-Holland, 1990.

[25] J.E. Hopcroft and R.E. Tarjan. Efficient algorithms for graph manipulation. *Communications fo the ACM*, 16(6), 1973.

[26] P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3), September 1989.

[27] J. Hughes. Why functional programming matters. *The Computer Journal*, 32(2), 1989.

[28] D.J. King and J. Launchbury. Structuring depth-first search algorithms in haskell. *Proceedings of the 22nd. ACM Symposium on Principles of Programming Languages, USA*, January 1995.

[29] J.H. Kingston. *Algorithms and Data Structures: Design, Correctness, Analysis.* Addison-Wesley, 1990.

[30] D.E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* ACM Press, Addison-Wesley, 1994.

[31] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7, 1956.

[32] G.R. Malcolm. *Algebraic Data Types and Program Transformation.* PhD thesis, Rijksuniversiteit Groningen, September 1990.

[33] L. Meertens. Constructing a calculus of programs. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction.* Springer-Verlag LNCS 375, 1989.

[34] L. Meertens. Paramorphisms. *Formal Aspects of Computing*, 4(5), 1992.

[35] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture.* Springer-Verlag LNCS 523, 1991.

[36] Q. Miller and P. Wadler. An introduction to Orwell 6. Oxford University Computing Laboratory, Programming Research Group, 1993.

[37] B. Möller. Derivation of graph and pointer algorithms. Report Nr. 280, Institus für Mathematik der Universität Augsburg, 1993.

[38] B. Möller and M. Russling. Shorter paths to graph algorithms. *Science of Computer Programming*, 22(1-2), April 1994.

[39] C. Morgan. *Programming from Specifications.* Prentice-Hall, 2nd edition, 1994.

[40] B.C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[41] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36, 1957.

[42] M. Russling. A general scheme for breadth-first graph traversal. In B. Möller, editor, *Mathematics of Program Construction.* Springer-Verlag LNCS 947, 1995.

[43] G. Schmidt and T. Ströhlein. *Relations and Graphs.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1993.

[44] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers and Mathematics with Applications*, 7(1), 1981.

[45] M. Spivey. A categorical approach to the theory of lists. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*. Springer-Verlag LNCS 375, 1989.

[46] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), 1972.

[47] K. Thulasiraman and M.N.S. Swamy. *Graphs: Theory and Algorithms*. John Wiley and Sons, 1992.

[48] M.A. Weiss. *Data Structures and Algorithm Analysis*. Benjamin / Cummings Publishing Company, 1992.

A The Thinning-the-Closure Rule

We first state yet another property of Θ that will be needed.

$$\Theta(B,A) \;=\; \Theta(B,A) - Q{\cdot}C \quad \text{provided} \;\; \begin{array}{l} A \cap Q{\cdot}C = \emptyset \\ \wedge\;\; C \subseteq B \end{array} \tag{A.1}$$

To prove it, we argue

$$\begin{array}{cl}
 & \Theta(B,A) \\
= & \quad \{ \text{ definition of } \Theta, \text{ fixed point } \} \\
 & A \;\cup\; (S{\cdot}\Theta(B,A) \;-\; Q{\cdot}B) \\
= & \quad \left\{ \begin{array}{l} \text{complements: } X = (X \cap Y) \cup (X - Y); \\ \text{second conjunct of proviso, join: } B = B \cup C \end{array} \right\} \\
 & (A \cap Q{\cdot}C) \;\cup\; (A - Q{\cdot}C) \;\cup\; (S{\cdot}\Theta(B,A) \;-\; Q{\cdot}(B \cup C)) \\
= & \quad \{ \text{ first conjunct of proviso; distribution } \} \\
 & (A - Q{\cdot}C) \;\cup\; (S{\cdot}\Theta(B,A) \;-\; Q{\cdot}B \;-\; Q{\cdot}C) \\
= & \quad \{ \text{ distribution } \} \\
 & (A \;\cup\; (S{\cdot}\Theta(B,A) \;-\; Q{\cdot}B)) \;-\; Q{\cdot}C \\
= & \quad \{ \text{ definition of } \Theta, \text{ fixed point } \} \\
 & \Theta(B,A) - Q{\cdot}C
\end{array}$$

Now for the big main task: the *thin*/Θ rule. In order to reduce the uses of converse in the proof, we restate the rule with $R := R^{\circ}$. It is valid to do so due to the isomorphism between preorders and converses of them. The new spelling out of the rule goes

$$\left.\begin{array}{l} \mathit{thin}\,(Q \cap R^{\circ}) \,\cdot\, \Lambda\,(\Theta(B,A) - Q{\cdot}C) \;\;\supseteq\;\; \Lambda\,\Theta(B \cup C, A - Q{\cdot}C) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{provided} \;\; \begin{array}{l} A{\cdot}C^{\circ} \subseteq R \\ S{\cdot}C - Q{\cdot}B \subseteq A \end{array} \end{array}\right\}$$

By universal property of *thin*, we have to prove

$$\Theta(B \cup C,\, A - Q{\cdot}C) \;\;\subseteq\;\; \Theta(B,A) - Q{\cdot}C \tag{a}$$
$$\Theta(B,A) - Q{\cdot}C \;\;\subseteq\;\; (Q \cap R) \cdot \Theta(B \cup C,\, A - Q{\cdot}C) \tag{b}$$

(a) takes only two steps:

$$\begin{array}{cl}
 & \Theta(B \cup C,\, A - Q{\cdot}C) \\
= & \quad \{ \; \Theta \text{ property (A.1) } \} \\
 & \Theta(B \cup C,\, A - Q{\cdot}C) \;-\; Q{\cdot}C \\
\subseteq & \quad \{ \text{ monotonicity properties of } \Theta \; \} \\
 & \Theta(B,A) - Q{\cdot}C
\end{array}$$

For (b), immediate intuition suggests that $Q{\cdot}C$ be moved to the right-hand side by universal property of substraction and Knaster-Tarski applied on $\Theta(B, A)$. Unfortunately, by so doing we arrive at a false statement. We pull out a rabbit that will permit a safe application of Knaster-Tarski. Note first that

$$X - Y = X - (Z \cap Y) \qquad \text{provided} \quad X \subseteq Z.$$

Using it to transform the left-hand side of (b) with a suitable Z allows us to proceed as intuition hinted initially. We use $Z := (Q \cap R) \cdot \Theta(B, A)$ –validly since $(Q \cap R)$ is reflexive–.

$$
\begin{array}{cll}
 & \text{(b)} & \\
\equiv & \left\{ \begin{array}{l} \text{introduce } V := \Theta(B,A),\ W := \Theta(B \cup C, A - Q{\cdot}C); \\ \text{rabbit described above} \end{array} \right\} & \\
 & V - ((Q \cap R){\cdot}V \cap Q{\cdot}C) \ \subseteq \ (Q \cap R){\cdot}W & \\
\equiv & \{ \text{ substraction } \} & \\
 & V \ \subseteq \ (Q \cap R){\cdot}W \ \cup \ ((Q \cap R){\cdot}V \cap Q{\cdot}C) & \\
\equiv & \{ \text{ introduce } X := (Q \cap R){\cdot}W,\ Y := (Q \cap R){\cdot}V \cap Q{\cdot}C \ \} & \\
 & V \ \subseteq \ X \cup Y & \\
\Leftarrow & \{ \text{ Knaster-Tarski } \} & \\
 & A \ \cup \ (S{\cdot}(X \cup Y) - Q{\cdot}B) \ \subseteq \ X \cup Y & \\
\equiv & \{ \text{ distribution, join } \} & \\
 & A \ \subseteq \ X \cup Y & \text{(b1)} \\
 & \wedge \ \ S{\cdot}X - Q{\cdot}B \ \subseteq \ X \cup Y & \text{(b2)} \\
 & \wedge \ \ S{\cdot}Y - Q{\cdot}B \ \subseteq \ X \cup Y & \text{(b3)}
\end{array}
$$

For (b1), we argue

$$
\begin{array}{cl}
 & A \\
= & \{ \text{ complements } \} \\
 & (A - Q{\cdot}C) \ \cup \ (A \cap Q{\cdot}C) \\
\subseteq & \left\{ \begin{array}{l} \text{by fixed point and join:} \\ \quad \Theta(J,K) \ = \ K \cup (S{\cdot}\Theta(J,K) - Q{\cdot}J) \ \supseteq \ K, \\ \text{hence } W \supseteq A - Q{\cdot}C \text{ and } V \supseteq A \end{array} \right\} \\
 & W \ \cup \ (V \cap Q{\cdot}C) \\
\subseteq & \{ \ (Q \cap R) \text{ reflexive, definitions of } X \text{ and } Y \ \} \\
 & X \cup Y
\end{array}
$$

For (b2), the left-hand side is first manipulated thus

$$\begin{array}{cl}
 & S{\cdot}X - Q{\cdot}B \\
\subseteq & \quad \{ \text{ definition of } X, \text{ (Hyp.2) } \} \\
 & (Q \cap R){\cdot}S{\cdot}W - Q{\cdot}B \\
\subseteq & \quad \left\{ \begin{array}{l} \text{by meet and } Q \text{ transitive: } (Q \cap R){\cdot}Q \subseteq Q, \\ \text{substraction anti-monotonic on right-hand operand} \end{array} \right\} \\
 & (Q \cap R){\cdot}S{\cdot}W - (Q \cap R){\cdot}Q{\cdot}B \\
\subseteq & \quad \{ \text{ substraction: } X{\cdot}Y - X{\cdot}Z \subseteq X{\cdot}(Y - Z) \ \} \\
 & (Q \cap R) \cdot (S{\cdot}W - Q{\cdot}B) \\
= & \quad \{ \text{ complements, distribution } \} \\
 & (Q \cap R) \cdot (S{\cdot}W - Q{\cdot}B - Q{\cdot}C) \ \cup \ (Q \cap R) \cdot ((S{\cdot}W - Q{\cdot}B) \cap Q{\cdot}C) \\
= & \quad \{ \text{ introduce } Z_1 \text{ and } Z_2 \ \} \\
 & Z_1 \cup Z_2
\end{array}$$

Hence, it suffices to show that $Z_1 \subseteq X$ and $Z_2 \subseteq Y$. The first half:

$$\begin{array}{cl}
 & Z_1 \\
\subseteq & \quad \left\{ \begin{array}{l} \text{by fixed point and join:} \\ \quad \Theta(J,K) = K \cup (S{\cdot}\Theta(J,K) - Q{\cdot}J) \supseteq S{\cdot}\Theta(J,K) - Q{\cdot}J, \\ \text{hence } W \supseteq S{\cdot}W - Q{\cdot}B - Q{\cdot}C \end{array} \right\} \\
 & (Q \cap R){\cdot}W \\
= & \quad \{ \text{ definition of } X \ \} \\
 & X
\end{array}$$

The second half:

$$\begin{array}{cl}
 & Z_2 \\
\subseteq & \quad \{ \text{ by monotonicity properties of } \Theta\text{: } W \subseteq V \ \} \\
 & (Q \cap R) \cdot ((S{\cdot}V - Q{\cdot}B) \cap Q{\cdot}C) \\
\subseteq & \quad \left\{ \begin{array}{l} \text{by fixed point and join:} \\ \quad \Theta(J,K) = K \cup (S{\cdot}\Theta(J,K) - Q{\cdot}J) \supseteq S{\cdot}\Theta(J,K) - Q{\cdot}J, \\ \text{hence } V \supseteq S{\cdot}V - Q{\cdot}B \end{array} \right\} \\
 & (Q \cap R) \cdot (V \cap Q{\cdot}C) \\
\subseteq & \quad \{ \text{ meet } \} \\
 & (Q \cap R){\cdot}V \ \cap \ (Q \cap R){\cdot}Q{\cdot}C \\
\subseteq & \quad \left\{ \begin{array}{l} \text{by meet and } Q \text{ transitive: } (Q \cap R){\cdot}Q \subseteq Q, \\ \text{definition of } Y \end{array} \right\} \\
 & Y
\end{array}$$

Lastly, for (b3), we proceed

$$
\begin{array}{cl}
 & S\cdot Y - Q\cdot B \\
= & \quad \{ \text{ definition of } Y \text{ } \} \\
 & S\cdot((Q\cap R)\cdot V \cap Q\cdot C) \,-\, Q\cdot B \\
\subseteq & \quad \{ \text{ modular law } \} \\
 & S\cdot((Q\cap R)\cdot V\cdot C^{\circ} \cap Q)\cdot C \,-\, Q\cdot B \\
\subseteq & \quad \{ \text{ } \Theta \text{ property (5.7), first proviso } \} \\
 & S\cdot((Q\cap R)\cdot R \cap Q)\cdot C - Q\cdot B \\
\subseteq & \quad \{ \text{ by meet and } R \text{ transitive: } (Q\cap R)\cdot R \subseteq R \text{ } \} \\
 & S\cdot(R\cap Q)\cdot C - Q\cdot B \\
\subseteq & \quad \{ \text{ (Hyp.2) } \} \\
 & (Q\cap R)\cdot S\cdot C - Q\cdot B \\
\subseteq & \quad \left\{ \begin{array}{l} \text{by meet and } Q \text{ transitive: } (Q\cap R)\cdot Q \subseteq Q, \\ \text{substraction anti-monotonic on right-hand operand} \end{array} \right\} \\
 & (Q\cap R)\cdot S\cdot C - (Q\cap R)\cdot Q\cdot B \\
\subseteq & \quad \{ \text{ substraction: } X\cdot Y - X\cdot Z \subseteq X\cdot(Y-Z) \text{ } \} \\
 & (Q\cap R)\cdot(S\cdot C - Q\cdot B) \\
\subseteq & \quad \{ \text{ second proviso } \} \\
 & (Q\cap R)\cdot A \\
\subseteq & \quad \{ \text{ (b1) proved above, distribution } \} \\
 & (Q\cap R)\cdot X \,\cup\, (Q\cap R)\cdot Y \\
\subseteq & \quad \left\{ \begin{array}{l} \text{definition of } X,\ (Q\cap R) \text{ transitive;} \\ \text{definition of } Y, \text{ meet} \end{array} \right\} \\
 & X \,\cup\, ((Q\cap R)\cdot V \cup (Q\cap R)\cdot Q\cdot C) \\
\subseteq & \quad \left\{ \begin{array}{l} \text{by meet and } Q \text{ transitive: } (Q\cap R)\cdot Q \subseteq Q, \\ \text{definition of } Y \end{array} \right\} \\
 & X\cup Y
\end{array}
$$

Done!

B Domains and Invariants

Formally, R is inductive if and only if

$$(\forall C : C \text{ coreflexive} : R \mathbin{\dot{\setminus}} C \subseteq C \;\Rightarrow\; id \subseteq C)$$

The coreflexive-division operator $\dot{\setminus}$ is defined as $R \mathbin{\dot{\setminus}} C = id \cap R \setminus (C \cdot \Pi)$.

Proposition 5.8 *Let R, S be relations and C a coreflexive such that R is entire, $C \subseteq dom\,(\mathsf{F}C \cdot S)$ and S is F-inductive. Then $C \subseteq dom\,(\mu X : R \cdot \mathsf{F}X \cdot S)$.*

Its proof follows:

$$\begin{array}{cl}
 & C \subseteq dom\,(\mu X : R \cdot \mathsf{F}X \cdot S) \\
\equiv & \quad \{ \text{ introduce } M := (\mu X : \ldots), \text{ division } \} \\
 & id \subseteq C \setminus dom\, M \\
\equiv & \quad \{ \text{ meet } \} \\
 & id \subseteq id \cap C \setminus dom\, M \\
\Leftarrow & \quad \{ \; S \;\mathsf{F}\text{-inductive, that is, } \delta \cdot S \text{ inductive } \} \\
 & (\delta \cdot S) \mathbin{\dot{\setminus}} (id \cap C \setminus dom\, M) \subseteq id \cap C \setminus dom\, M \\
\equiv & \quad \{ \; X \mathbin{\dot{\setminus}} D \text{ coreflexive, meet, division } \} \\
 & C \cdot (\delta \cdot S) \mathbin{\dot{\setminus}} (id \cap C \setminus dom\, M) \subseteq dom\, M
\end{array}$$

And for the last inequation we argue

$$\begin{array}{cl}
 & C \cdot (\delta \cdot S) \mathbin{\dot{\setminus}} (id \cap C \setminus dom\, M) \\
= & \quad \{ \text{ a coreflexive equals its domain } \} \\
 & dom\,(C \cdot (\delta \cdot S) \mathbin{\dot{\setminus}} (id \cap C \setminus dom\, M)) \\
\subseteq & \quad \left\{ \begin{array}{l} \text{proviso on } C \text{ and } S \text{, and} \\ \text{domains: } dom\,(dom\, X \cdot Y) = dom\,(X \cdot Y) \end{array} \right\} \\
 & dom\,(\mathsf{F}C \cdot S \cdot (\delta \cdot S) \mathbin{\dot{\setminus}} (id \cap C \setminus dom\, M)) \\
= & \quad \{ \text{ coreflexive division: } (\delta \cdot X) \mathbin{\dot{\setminus}} D = X \mathbin{\dot{\setminus}} \mathsf{F}D \; \} \\
 & dom\,(\mathsf{F}C \cdot S \cdot S \mathbin{\dot{\setminus}} \mathsf{F}(id \cap C \setminus dom\, M)) \\
= & \quad \{ \text{ coreflexive division: } X \cdot X \mathbin{\dot{\setminus}} D \subseteq D \cdot X \; \} \\
 & dom\,(\mathsf{F}C \cdot \mathsf{F}(id \cap C \setminus dom\, M) \cdot S) \\
\subseteq & \quad \{ \text{ meet, monotonic functors, division } \} \\
 & dom\,(\mathsf{F}(dom\, M) \cdot S) \\
\subseteq & \quad \left\{ \begin{array}{l} \text{domains and monotonic} \\ \text{functors: } \mathsf{F}(dom\, X) \subseteq dom\, \mathsf{F}X \end{array} \right\} \\
 & dom\,(dom\, \mathsf{F}M \cdot S)
\end{array}$$

$$
\begin{array}{cl}
= & \{ \text{ domains } \} \\
& \mathit{dom}\,(\mathsf{F}M \cdot S) \\
= & \{ \text{ } R \text{ entire } \} \\
& \mathit{dom}\,(R \cdot \mathsf{F}M \cdot S) \\
= & \{ \text{ definition of } M\text{, fixed point } \} \\
& \mathit{dom}\, M.
\end{array}
$$

Proposition 5.9 *If C is F-invariant under S, then*

$$(\mu X : R \cdot \mathsf{F}X \cdot S) \cdot C \quad = \quad (\mu X : R \cdot \mathsf{F}X \cdot S \cdot C)$$

The proof is by mutual inclusion. Name M_1, M_2 the least fixed point expressions on the left and right-hand side respectively. We start with $(\supseteq)$:

$$
\begin{array}{cl}
& M_2 \\
= & \{ \text{ definition of } M_2\text{, fixed point } \} \\
& R \cdot \mathsf{F}M_2 \cdot S \cdot C \\
= & \{ \text{ duplicity of coreflexives, fixed point back again } \} \\
& M_2 \cdot C \\
\subseteq & \{ \text{ } C \text{ coreflexive, monotonicity of } \mu \text{ } \} \\
& M_1 \cdot C
\end{array}
$$

$(\subseteq)$ takes a little longer:

$$
\begin{array}{cl}
& M_1 \cdot C \subseteq M_2 \\
\equiv & \{ \text{ division } \} \\
& M_1 \subseteq M_2/C \\
\Leftarrow & \{ \text{ definition of } M_1\text{, Knaster-Tarski } \} \\
& R \cdot \mathsf{F}(M_2/C) \cdot S \subseteq M_2/C \\
\equiv & \{ \text{ division } \} \\
& R \cdot \mathsf{F}(M_2/C) \cdot S \cdot C \subseteq M_2
\end{array}
$$

And the last line is proved thus

$$
\begin{array}{cl}
& R \cdot \mathsf{F}(M_2/C) \cdot S \cdot C \\
\subseteq & \{ \text{ duplicity of coreflexives, } C \text{ } \mathsf{F}\text{-invariant under } S \text{ } \} \\
& R \cdot \mathsf{F}(M_2/C) \cdot \mathsf{F}C \cdot S \cdot C \\
\subseteq & \{ \text{ monotonic functors, division } \} \\
& R \cdot \mathsf{F}M_2 \cdot S \cdot C \\
= & \{ \text{ definition of } M_2\text{, fixed point } \} \\
& M_2.
\end{array}
$$