A calculational theory of pers as types

Graham Hutton, University of Glasgow
graham@dcs.glasgow.ac.uk

Ed Voermans, Eindhoven University of Technology
wsinedv@win.tue.nl

Appears as research report 1992/R1,
Department of Computing Science, University of Glasgow, 1992

January 1992

Abstract

We present a programming paradigm based upon the notion of binary relations as programs, and partial equivalence relations (pers) as types. Our method is *calculational*, in that programs are derived from specifications by algebraic manipulation. Working with relations as programs generalises the functional paradigm, admiting non–determinism and the use of relation converse. Working with pers as types, we have a more general notion than normal of what constitutes an element of a type; this leads to a more general class of functional relations, the so–called *difunctional* relations. Our basic method of defining types is to take the fixpoint of a *relator*, a simple strengthening of the categorical notion of a functor. Further new types can be made by imposing laws and restrictions on the constructors of other types. Having pers as types is fundamental to our treatment of types with laws.

Contents

Chapter 1

Introduction

Perhaps the best known calculational paradigm is the Bird–Meertens formalism, or to use its more colloquial name, Squiggol [1]. Programs in the Squiggol style work upon trees, lists, bags and sets, the so–called Boom hierarchy. The framework was uniformly extended to cover arbitrary recursive types by Malcolm in [2], by means of the F–algebra paradigm of type definition, and resulting catamorphic programming style. More recently, Backhouse et al [3] have made a further generalisation by choosing to work within the relational calculus.

Choosing to work with relations rather than functions allows non–determinism to be incorporated into our calculations. Programs in the Squiggol style are total functions, producing precisely one output for each input. Relational programs may produce an arbitrary number of results for each input, or indeed, none at all; many programming problems are inherently non–deterministic in nature.

Of fundamental importance to Backhouse's relational approach is the treatment of types not as separate entities from programs, but as special kinds of programs. Specifically, partial identity relations, called monotypes in [3], are adopted as types. Working in this style allows programs and type information to be manipulated withing the single framework of the relational calculus, and is fundamental to the use of type information to "inform and structure" program calculation. We in this paper take a more general approach than Backhouse, adopting partial equivalence relations as types. The extra generality allows us to handle types whose constructors are required to satisfy equations, so–called "types with laws".

An earlier version of this article appears as [4]; an excerpt presenting difunctional relations appears as [5]. Before moving on, we make a few remarks:

Notation

We in this article follow the notation and terminology as used by Backhouses in [3]. Rather than working with the normal "$\subseteq$", "$\cup$", and "$\cap$" operators from set–theory, Backhouse adopts in his axiomatic framework the more anonymous lattice–theoretic symbols "$\sqsubseteq$", "$\sqcup$" and "$\sqcap$". The term *spec* (abbreviating specification) is used in preference to *relation*, keeping a clear distinction between the axiomatic theory and the meta–language (predicate calculus.) Throughout this article capital letters

$R, S, T, \ldots$ are used to denote arbitrary specs. To avoid confusion with the use of the capital letter "T" to denote an arbitrary spec, the supreme spec (with respect to the implicit universe) is denoted by "$\top\!\!\top$" rather than the more usual lattice theoretic symbol "$\top$". The empty–spec is written as "$\bot\!\!\bot$". The composition operator for specs is denoted by "$\circ$", and the identity spec with respect to this operator by "I". For relation converse (an operator which swaps the components of each pair in a binary relation) the anonymous notation $R^{\cup}$ is used rather than the more usual R^{-1}; writing R^{-1} might have led us to suspect that R^{-1} is the inverse of R with respect to the composition operator "$\circ$", which is not true in general.

Proofs

We adopt the proof style put forward by Dijkstra, Feijen, van Gasteren, and others. (See for example [6] and [7].) The reader should have little problem in understanding our calculations without any prior knowledge of this style.

We have two comments to make about the proof style. First of all, monotonicity properties of the operators of relational calculus are used so frequently that they are mostly not mentioned in the hints. Secondly, the prolific use of reverse implication "$\Leftarrow$" rather than the perhaps more familiar "$\Rightarrow$" reflects the direction in which proofs are most often made: in proving $A \Rightarrow B$, experience has shown that the process is often much simpler if we begin with the consequent B, and work backwards to the assumption A. Thus, each proof step in this style is either an equivalence, or a reverse implication. When more than one assumption A is involved, it is convenient not to work backwards directly to all of them, but include assumptions as "hints" at some stage in the proof, working ultimately towards the empty assumption *true*.

Quantification

For quantified expressions, we adopt the *Eindhoven notation* [7]. The general pattern is $Q\ (x : p.x : t.x)$, where Q is some quantifier (e.g. "$\forall$" or "$\bigsqcup$"), x is a sequence of free variables (sometimes called *dummies*), $p.x$ is a predicate which must be satisfied by the dummies, and $t.x$ is an expression defined in terms of the dummies. For the special cases of "$\forall$" and "$\exists$", we have the following links with standard notation:

$$\forall\ (x : p.x : q.x) \;\equiv\; \forall x.\ (p.x \Rightarrow q.x)$$

$$\exists\ (x : p.x : q.x) \;\equiv\; \exists x.\ (p.x \wedge q.x)$$

Chapter 2

Relational calculus

Our per–based paradigm is developed within the "axiomatic theory of relations" as presented in [3]. To paraphrase Backhouse: "We prefer the axiomatic approach to working directly with set–theoretic relations because of our concerns with clear and economical calculation; formal manipulations involving bound variables and quantifiers quickly become long and unwieldy, and can obscure rather than elucidate an argument. The minimisation of bound variables has long been advocated by category theory, as well as being fundamental to the Squiggol paradigm."

We present in this chapter an overview of Backhouse's axiomatic approach to the relational calculus. It comprises three layers, dealing with the powerset lattice structure of relations, the composition operator "$\circ$", and the reverse operator "$\cup$". We also quickly review the cone axiom, factors, and monotypes. (For all proofs omitted in this chapter, the reader is referred to [3].)

For reference we give below pointwise definitions for the operators as used in this article. For R a binary relation, $x\ R\ y$ may be read as "x is related by R to y." Formally then, $x\ R\ y$ is just an abbreviation for $(x, y) \in R$.

$$
\begin{array}{lcl}
x \perp\!\!\perp y & \widehat{=} & \mathit{false} \\
x \top\!\!\top y & \widehat{=} & \mathit{true} \\
R \sqsupseteq S & \widehat{=} & \forall\,(x, y :: x\ R\ y \ \Leftarrow\ x\ S\ y) \\
x\ (R \sqcup S)\ y & \widehat{=} & x\ R\ y\ \vee\ x\ S\ y \\
x\ (R \sqcap S)\ y & \widehat{=} & x\ R\ y\ \wedge\ x\ S\ y \\
x\ (\neg R)\ y & \widehat{=} & \sim(x\ R\ y) \\
x\ (R \circ S)\ z & \widehat{=} & \exists\,(y :: x\ R\ y\ \wedge\ y\ S\ z) \\
x\ I\ y & \widehat{=} & x = y \\
x\ (R^{\cup})\ y & \widehat{=} & y\ R\ x
\end{array}
$$

2.1 Powerset lattice structure

Recall that a *lattice* is a partially ordered set for which "lubs" and "glbs" exist of each finite subset. (Recall that lub abbreviates *least upper bound*, while glb abbreviates *greatest lower bound.*) Existence of finite lubs and glbs is known to be equivalent to the existence of least and greatest elements, denoted respectively by "$\perp\!\!\perp$" and "$\top\!\!\top$", and binary lub and glb operators, characterised uniquely by

$$Z \sqsupseteq X \wedge Z \sqsupseteq Y \quad \equiv \quad Z \sqsupseteq X \sqcup Y$$

$$X \sqsupseteq Z \wedge Y \sqsupseteq Z \quad \equiv \quad X \sqcap Y \sqsupseteq Z$$

Thus, $X \sqcup Y$ is the least Z subsuming both X and Y, while $X \sqcap Y$ is the greatest Z approximating both X and Y. We note that both "$\sqcup$" and "$\sqcap$" are idempotent, commutative, associative, and that $\perp\!\!\perp/\top\!\!\top$ are the identity elements for $\sqcup/\sqcap$ respectively; conversely, $\top\!\!\top/\perp\!\!\perp$ are zero elements for $\sqcup/\sqcap$.

A lattice is called *complete* when lubs and glbs of all infinite sets of elements also exist. (It may be surprising to note that the existence of infinite lubs is equivalent to the existence of infinite glbs; postulating the existence of either implies the other.)

Lattice elements X and Y are called *complements* if

$$X \sqcup Y = \top\!\!\top \quad \wedge \quad X \sqcap Y = \perp\!\!\perp$$

A lattice is called *complemented* if every element X has a unique such complement, which we choose to write as $\neg X$, and pronounce "non X".

A lattice is called *distributive* if the lub operator distributes over finite glbs, and the glb operator distributes over finite lubs. These two finite distribution properties are easily seen to be equivalent to two binary distribution properties:

$$X \sqcup (Y \sqcap Z) \; = \; (X \sqcup Y) \sqcap (X \sqcup Z)$$

$$X \sqcap (Y \sqcup Z) \; = \; (X \sqcap Y) \sqcup (X \sqcap Z)$$

These two laws are in fact equivalent, e.g. the first follows from the second:

$$
\begin{array}{ll}
 & (X \sqcup Y) \sqcap (X \sqcup Z) \\
= & \qquad \{ \text{ distribution } \} \\
 & ((X \sqcup Y) \sqcap X) \;\sqcup\; ((X \sqcup Y) \sqcap Z) \\
= & \qquad \{ \text{ absorbtion } \} \\
 & X \;\sqcup\; ((X \sqcup Y) \sqcap Z) \\
= & \qquad \{ \text{ distribution } \} \\
 & X \sqcup (X \sqcap Z) \sqcup (Y \sqcap Z) \\
= & \qquad \{ \text{ absorbtion } \} \\
 & X \sqcup (Y \sqcap Z)
\end{array}
$$

In a complete lattice, if the lub and glb operators distribute over glbs and lubs of all infinite sets of elements, the lattice is called *completely distributive.*

The first layer of our axiom system for the relational calculus is defined in terms of the lattice–theoretic notions introduced above. We begin with some fixed set $\mathcal{A}$, on which we impose the structure of a complete, completely distributive complemented lattice. Given any set $\mathcal{X}$, the power–set $\wp(\mathcal{X})$ forms such a structure; for this reason, the structure is often called a power–set lattice, or *plat* for short.

Being defined in terms of the powerset over a product space, we see that set–theoretic relations over some universe form such a plat structure. Rather than referring to elements of $\mathcal{A}$ as "relations", Backhouse [3] prefers to keep a clear distinction between the meta–language (predicate calculus) and the axiomatic theory itself, choosing to refer to elements of $\mathcal{A}$ under the special name of *specs* (abbreviating "specifications.") In keeping with the link between specs and relations, throughout the paper we use the capital letters $R, S, T, \ldots$ to denote arbitrary specs. In combination, we refer to the three layers of our axiom system as the *spec calculus*.

(We might have expected types to enter the picture at this stage. Specs at present are untyped; we return to this point in the last part of this introduction.)

An endo–function f on the plat structure $\mathcal{A}$ is called *monotonic* iff

$$f.X \sqsupseteq f.Y \;\Leftarrow\; X \sqsupseteq Y$$

Among the more significant properties of plat structures is their satisfaction of the "Knaster–Tarski fixpoint theorem" [8]. For an arbitrary monotonic function f, this theorem tells us that the equation $X = f.X$ has a least solution for X, denoted by μF, and characterised uniquely by the following two properties:

$$f.\mu f = \mu f \quad \wedge \quad (X \sqsupseteq \mu f \;\Leftarrow\; X \sqsupseteq f.X)$$

We refer to the first conjunct above as the *calculation rule* for μf, and the second conjunct as the *fixpoint induction rule* for μf. We might have expected the second conjunct to take the form "$X \sqsupseteq \mu f \;\Leftarrow\; X = f.X$" here, expressing directly that the least fixpoint is weaker than any other. It is however known that the least solutions to the equations $X = f.X$ and $X \sqsupseteq f.X$ coincide, so we are free to weaken the antecedent of fixpoint induction rule to a containment. The Knaster–Tarski theorem predicts not just a least solution for the equation $X = f.X$, but in fact a lattice of solutions. We denote the greatest such solution by νf. In general, if F is a family of monotonic functions, the simultaneous inclusions $(f : f \in F : X \sqsupseteq f.X)$ have a lattice of solutions in X; the least such X satisfies $X = \bigsqcup(f : f \in F : f.X)$.

2.2 Composition

The second layer of the axiom system is the monoid structure for composition of specs, where "$\circ$" is an associative binary infix operator, with unit element "I".

Axiom 1: $R \circ (S \circ T) \;=\; (R \circ S) \circ T$

Axiom 2: $I \circ R = R = R \circ I$

The interface between the plat and monoid layers is that "$\circ$" is pointwise universally lub–junctive: for all sets X and Y of specs,

Axiom 3: $(\bigsqcup X) \circ (\bigsqcup Y) \;=\; \bigsqcup(x, y : x \in X \;\wedge\; y \in Y : x \circ y)$

This axiom is most often used in the following two forms, stating respectively that $(R\,\circ)$ and $(\circ\, R)$ distribute over the binary "$\sqcup$" operator.

$$R \circ (S \sqcup T) \;=\; R \circ S \,\sqcup\, R \circ T$$

$$(S \sqcup T) \circ R \;=\; S \circ R \,\sqcup\, T \circ R$$

From the junctivity axiom, we can derive some familiar laws:

- $\perp\!\!\perp$ is a zero for composition: $\perp\!\!\perp \circ R \;=\; R \circ \perp\!\!\perp \;=\; \perp\!\!\perp$
- composition is monotonic in both arguments
- $\top\!\!\top \circ \top\!\!\top \;=\; \top\!\!\top$

What now about a related axiom for glbs, saying that "$\circ$" is pointwise universally glb–junctive: $(\sqcap X) \circ (\sqcap Y) \;=\; \sqcap\,(x, y : x \in X \;\wedge\; y \in Y : x \circ y)$? For set–theoretic relations, it does not hold true in general. Indeed, if it were present, our theory would collapse to a 1–point system: under the assignments $X := \emptyset$ and $Y := \{\perp\!\!\perp\}$, the glb–junctivity equation holding true requires that $\perp\!\!\perp \,=\, \top\!\!\top$.

2.3 Relation converse

The third and final layer of the axiom system is the reverse structure, where "$\cup$" (pronounced colloquially as "wok") is a postfix operator that is its own inverse:

Axiom 4: $(R^{\cup})^{\cup} = R$

The interface with the plat layer is that "$\cup$" both respects and preserves the plat ordering, thus being an isomorphism between the plat $\mathcal{A}$ and itself:

Axiom 5: $R \sqsupseteq S \;\equiv\; R^{\cup} \sqsupseteq S^{\cup}$

Two remarks on notation: Firstly, we choose "$\cup$" as a postfix operator because of its commutativity with "$\neg$" (as noted below). As a result, this law takes the form $(\neg R)^{\cup} = \neg(R^{\cup})$, allowing us to write $\neg R^{\cup}$ without bracketing being required. In this manner, the commutativity law need never be explicitly applied in calculation, it being handled silently by our careful choice of notation. Secondly, we prefer the anonymous notation $R^{\cup}$ to the more usual R^{-1}, which might lead us to suspect that R^{-1} is the inverse of R with respect to "$\circ$", which is not true in general.

The previous two axioms give the link between the "$\cup$" operator and itself, and "$\cup$" and the "$\sqsupseteq$" ordering; the expected links between "$\cup$" and the remaining five operators of the plat structure follow directly from these two axioms:

- $\perp\!\!\perp^{\cup} = \perp\!\!\perp$
- $\top\!\!\top^{\cup} = \top\!\!\top$
- $(R \sqcup S)^{\cup} \;=\; R^{\cup} \sqcup S^{\cup}$
- $(R \sqcap S)^{\cup} \;=\; R^{\cup} \sqcap S^{\cup}$
- $\neg(R^{\cup}) = (\neg R)^{\cup}$

The interface between the reverse and composition layers is that "$\cup$" distributes contravariantly over "$\circ$" (i.e. note the reversal of R and S below.)

Axiom 6: $(R \circ S)^{\cup} \;=\; S^{\cup} \circ R^{\cup}$

We might expect similarly that $I^{\cup} = I$ be required as an axiom. As shown in the calculation below however, this identity in fact follows automatically from other axioms. (We see now that "$\cup$" is a contravariant monoid endomorphism.)

$$
\begin{array}{ll}
 & I^{\cup} \\
= & \quad \{\ \text{identity}\ \} \\
 & I^{\cup} \circ I \\
= & \quad \{\ \text{reverse}\ \} \\
 & (I^{\cup} \circ I)^{\cup\cup} \\
= & \quad \{\ \text{distribution}\ \} \\
 & (I^{\cup} \circ I^{\cup\cup})^{\cup} \\
= & \quad \{\ \text{reverse}\ \} \\
 & (I^{\cup} \circ I)^{\cup} \\
= & \quad \{\ \text{identity}\ \} \\
 & I^{\cup\cup} \\
= & \quad \{\ \text{reverse}\ \} \\
 & I
\end{array}
$$

2.4 Cone axiom

The equivalence $\top\!\!\top \circ R \circ \top\!\!\top = \perp\!\!\perp \;\equiv\; R = \perp\!\!\perp$ holds in the spec calculus [3]. The related equivalence $\top\!\!\top \circ R \circ \top\!\!\top = \top\!\!\top \;\equiv\; R \neq \perp\!\!\perp$ holds in the standard set–theoretic model, provided that $\perp\!\!\perp \neq \top\!\!\top$; even under this assumption, the equivalence does not hold for all models of the spec calculus. It is dubbed the *cone axiom*.

Axiom 7: $\top\!\!\top \circ R \circ \top\!\!\top = \top\!\!\top \;\equiv\; R \neq \perp\!\!\perp$

The presence of this axiom precludes trivial models of the spec calculus. For example, substituting $R := \top\!\!\top$ gives $\perp\!\!\perp \neq \top\!\!\top$. More generally, the cone axiom prevents the assignment $(\cup, \circ, I) := (id, \sqcap, \top\!\!\top)$, which would otherwise satisfy the spec axioms.

2.5 Factors

A somewhat unfamiliar aspect of the spec calculus is the existence of two *factoring* operators, which as we shall see, may be thought of as approximate inverses to composition. Consider the following inclusions in X and Y:

$$R \sqsupseteq X \circ S$$

$$S \sqsupseteq R \circ Y$$

Least solutions are trivial: $X, Y := \perp\!\!\perp$. Least non–bottom solutions do not exist in general: any singleton subset of a solution is, by monotonicity of composition, also itself a solution; the only smaller than all of these is $\perp\!\!\perp$ itself.

Interesting greatest solutions for X and Y above do however exist, which we denote respectively by R/S (pronounced "R over S") and $R\backslash S$ (pronounced "R under S"). Formally speaking, the "/" and "\" operators are defined by

Definition 8:

$$R/S \;\;\widehat{=}\;\; \neg(\neg R \circ S^\cup)$$

$$R\backslash S \;\;\widehat{=}\;\; \neg(R^\cup \circ \neg S)$$

In practice however, the following properties prove much more useful:

Lemma 9:

$$R \sqsupseteq X \circ S \;\equiv\; R\,/\,S \sqsupseteq X$$

$$S \sqsupseteq R \circ Y \;\equiv\; R\,\backslash\,S \sqsupseteq Y$$

These equivalences are instances of so–called "Galois connections". Other examples are the definitions of "$\sqcup$" and "$\sqcap$" given earlier. From our point of view, each such equivalence encapsulates two facts, firstly that the operator satisfies some property (in this case, yielding a solution to an inclusion), and secondly, that these solutions are greatest (or least) such. As we shall see, combining these two facts in a single equivalence is of great benefit when calculating with such operators.

Putting $X := R/S$ and $Y := R\backslash S$ yields a pair of *cancellation laws*:

Lemma 10:

$$R \sqsupseteq R\,/\,S \circ S$$

$$S \sqsupseteq R \circ R\,\backslash\,S$$

It is these laws which motivate our choice of notation: in each case the cancelled expressions are adjacent, a useful aid to memory.

Putting $R, X := R \circ S, R$ and $S, Y := R \circ S, S$ yields another such pair:

Lemma 11:

$$(R \circ S) \;/\; S \;\sqsupseteq\; R$$

$$R \;\backslash\; (R \circ S) \;\sqsupseteq\; S$$

The four cancellation laws express an important relationship between composition, and the operators "/" and "\". Let us consider, for example, the two cancellation laws for "/"; abstracting on the spec R in both cases gives:

$$id \sqsupseteq (\circ S) \circ (/S)$$

$$(/S) \circ (\circ S) \sqsupseteq id$$

These laws tell us that $(\circ S)$ and $(/S)$ are approximate inverses in the spec calculus, in much the same way that $(*\ B)$ and $(div\ B)$ are approximate inverses in arithmetic. (The similarity is not exact, since in arithmetic we have the identity $(A * B)\ div\ B = A$, rather than just the containment $(A * B)\ div\ B \geq A$.)

In arithmetic, arguments to "$*$" are called *factors*. In keeping with the similarity between "$\circ$" and "$*$", and between "/" and "*div*", we call R/S a *left factor* of R, and $R \backslash S$ a *right factor* of S. The use of the qualifiers "left" and "right" reflects the position of R/S and $R \backslash S$ with respect to "$\circ$" in the first pair of cancellation laws. (Memory aid: in writing the "/" symbol in the 'left' factor R/S, our pen moves downwards to the 'left', while for 'right' factor $R \backslash S$, it moves to the 'right'.)

Factors have applications in many areas. Of particular relevance to the present work is Hoare and He's notion of "weakest pre– and post–specifications" [9]. (The reader should be aware that the notation used in [9] is precisely opposite to ours: e.g. they would write S/R where we write $R \backslash S$.) Despite their important calculational benefits (particularly in avoiding use of the "$\neg$" operator), the factoring operators are not widely known, being used in few texts on relational calculus.

2.6 Monotypes and domain operators

In the next chapter, we introduce our notion of types: "partial equivalence relations". Backhouse in [3] works with less general types: "partial identity relations". Following Backhouse, we refer to partial identity relations as *monotypes*.

Definition 12: spec A is a *monotype* $\;\;\widehat{=}\;\; I \sqsupseteq A$

To aid reading of formulae, monotypes are always denoted by letters A, B, C, etc. Set–theoretically, a monotype is just the identity relation on some set. For example, $\{(\mathsf{false}, \mathsf{false}), (\mathsf{true}, \mathsf{true})\}$ is a monotype representing a type of booleans. We list below a few simple properties of monotypes.

- $R \sqsupseteq A \circ R$
- $A = A^{\cup} = A \circ A$
- $A \circ B = A \sqcap B$

We often need to refer to the domain and range of a spec. In order to avoid confusion with decision to have the input part of functional relations on their right–side (explained later on in this article), we use the terms *left domain* and *right domain*. In the case of functional relations then, the right domain is the input part. In the spec calculus, a left domain of a spec R is represented by a monotype A satisfying $A \circ R = R$; similarly a right domain of R is represented by a monotype B satisfying $R \circ B = R$. Specs in general have many domains; the smallest left and right domains are given by operators "<" and ">", defined as follows:

Definition 13:

$$R< \ \widehat{=} \ I \sqcap (R \circ R^{\cup})$$

$$R> \ \widehat{=} \ I \sqcap (R^{\cup} \circ R)$$

Often more useful in practice are two equivalent formulations:

Lemma 14:

$$R< \ = \ I \sqcap (R \circ \top\!\!\top)$$

$$R> \ = \ I \sqcap (\top\!\!\top \circ R)$$

Using "<" and ">" we can define what it means for two specs to be disjoint:

Definition 15: $disj.R.S \ \widehat{=} \ R< \sqcap S< \ = \ R> \sqcap S> \ = \ \bot\!\!\bot$

The domain operators can in fact be eliminated:

Lemma 16: $disj.R.S \ \equiv \ R \circ S^{\cup} \ = \ R^{\cup} \circ S \ = \ \bot\!\!\bot$

We conclude with an simple lemma, used many times in this article. For want of a more appropriate name, Lemma 17 is dubbed the *triple rule*.

Lemma 17: $R \circ R^{\cup} \circ R \sqsupseteq R$

Proof

$$\begin{array}{ll}
 & R \circ R^{\cup} \circ R \\
\sqsupseteq & \quad \{ \ R^{\cup} \circ R \sqsupseteq R> \ \} \\
 & R \circ R> \\
= & \quad \{ \ \text{domains} \ \} \\
 & R
\end{array}$$

Chapter 3

Pers as types

A *partial equivalence relation* (per) on a set $\mathcal{S}$ is a symmetric and transitive relation on $\mathcal{S}$. To aid reading of formulae, we always use letters A, B and C for pers. In the spec–calculus, we have three equivalent definitions for the *per* notion.

Definition 18: $per.A \;\; \widehat{=}$

(a) $A = A^{\cup} \;\; \wedge \;\; A \sqsupseteq A \circ A$

(b) $A = A^{\cup} \;\; \wedge \;\; A = A \circ A$

(c) $A = A \circ A^{\cup}$

The first definition is most often used when we are required to show that A is a per, while the second is normally most convenient when we use that A is a per. (Equalities are in general more convenient than inclusions for calculation.) The third definition has the advantage of being the smallest, and sometimes proves useful.

Consider for example definition 18a. The calculation below shows that the $A \sqsupseteq A \circ A$ part corresponds to the set–theoretic notion of transitivity; a similar calculation would show that $A = A^{\cup}$ corresponds to the symmetry condition.

$$
\begin{array}{ll}
 & A \sqsupseteq A \circ A \\
\equiv & \qquad \{ \text{ def } \sqsupseteq \; \} \\
 & \forall\,(x,z :: \; x \; A \; z \Leftarrow x \; A{\circ}A \; z) \\
\equiv & \qquad \{ \text{ def } \circ \; \} \\
 & \forall\,(x,z :: \; x \; A \; z \Leftarrow \exists\,(y :: x \; A \; y \wedge y \; A \; z)) \\
\equiv & \qquad \{ \; \exists \text{ elimination } \} \\
 & \forall\,(x,y,z :: \; x \; A \; z \; \Leftarrow \; x \; A \; y \wedge y \; A \; z)
\end{array}
$$

Let us now show equivalence between the definitions 18a and 18b. (Equivalence with 18c is a simple exercise in manipulating pers, and is left to the reader.) Under the assumption that A is symmetric, i.e. $A = A^{\cup}$, we are required to show that the transitivity condition $A \sqsupseteq A \circ A$ is equivalent to $A = A \circ A$. Since the equality implies the containment, the following proof is sufficient:

$$
\begin{array}{ll}
 & A \circ A \\
\sqsupseteq & \quad \{ \text{ assumption: } A \sqsupseteq A \circ A \ \} \\
 & A \circ A \circ A \\
= & \quad \{ \text{ assumption: } A = A^{\cup} \ \} \\
 & A \circ A^{\cup} \circ A \\
\sqsupseteq & \quad \{ \text{ triple rule } \} \\
 & A
\end{array}
$$

A per on a set $\mathcal{S}$ that is reflexive on $\mathcal{S}$ is called an *equivalence relation* on $\mathcal{S}$. We use the notation $\mathcal{S}^2$ for the *full relation* $\mathcal{S} \times \mathcal{S}$ on $\mathcal{S}$. Formally then we have $x\ \mathcal{S}^2\ y \ \ \widehat{=}\ \ x, y \in S$. For example, $\{a, b\}^2 = \{(a, a), (a, b), (b, a), (b, b)\}$. It is clear that $\mathcal{S}^2$ is an equivalence relation on $\mathcal{S}$, indeed it is the largest such. The set–theoretic notion of a full relation can be cast in the spec calculus as follows:

> **Definition 19:** spec X is a *full relation* $\ \widehat{=}\ \ X = X \circ \top\!\!\top \circ X^{\cup}$.

We always use letters X, Y and Z for full relations. A well known result is:

> **Lemma 20:** every per can be written in a unique way as the union of disjoint full relations, each such relation representing an *equivalence class* of the per.

With "pers as types", it is these disjoint full relations that we refer to as *elements* of types. Aiming towards defining an "$\in$" operator for pers, we see that two properties of a set $\mathcal{S}$ are required such that $\mathcal{S}^2$ may be called an element of a per A:

(1) $\forall\,(x, y \ :\ x, y \in \mathcal{S} \ :\ x\ A\ y)$

(2) $\forall\,(x, y :\ x \in \mathcal{S} \ \wedge\ x\ A\ y \ :\ y \in \mathcal{S})$

The first clause says that $\mathcal{S}$ is an equivalence class of A, being just the set–theoretic expansion of $\mathcal{S}^2 \sqsubseteq A$. The second clause ensures that $\mathcal{S}$ is as big as possible. The following calculation shows that (2) is equivalent to $\mathcal{S}^2 \circ A \sqsubseteq \mathcal{S}^2$.

$$
\begin{array}{ll}
 & \mathcal{S}^2 \circ A \sqsubseteq \mathcal{S}^2 \\
\equiv & \quad \{ \text{ def } \sqsubseteq \ \} \\
 & \forall\,(x, z :\ z\ \mathcal{S}^2{\circ}A\ x :\ z\ \mathcal{S}^2\ x) \\
\equiv & \quad \{ \text{ def } \circ \ \} \\
 & \forall\,(x, y, z :\ z\ \mathcal{S}^2\ y \wedge y\ A\ x :\ z\ \mathcal{S}^2\ x) \\
\equiv & \quad \{ \text{ def } \mathcal{S}^2 \ \} \\
 & \forall\,(x, y, z :\ y, z \in \mathcal{S} \wedge y\ A\ x :\ x, z \in \mathcal{S}) \\
\equiv & \quad \{ \text{ calculus } \} \\
 & \forall\,(x, y, z :\ y, z \in \mathcal{S} \wedge y\ A\ x :\ x \in \mathcal{S}) \\
\equiv & \quad \{ \ \forall\,(x, y :\ y \in \mathcal{S} \wedge y\ A\ x :\ \exists\,(z \ ::\ z \in \mathcal{S})) \ \} \\
 & \forall\,(x, y :\ y \in \mathcal{S} \wedge y\ A\ x :\ x \in \mathcal{S})
\end{array}
$$

Given $\mathcal{S}^2 \sqsubseteq A$ we find that $\mathcal{S}^2 \circ A \sqsubseteq \mathcal{S}^2$ is equivalent to the equality $\mathcal{S}^2 \circ A = \mathcal{S}^2$. Since the equality implies the containment, the following proof is sufficient:

$$
\begin{array}{ll}
 & \mathcal{S}^2 \circ A \\
\sqsupseteq & \quad \{ \text{ assumption: } A \sqsupseteq \mathcal{S}^2 \ \} \\
 & \mathcal{S}^2 \circ \mathcal{S}^2 \\
= & \quad \{ \text{ full relations } \} \\
 & \mathcal{S}^2
\end{array}
$$

We give now a spec–calculi definition for an "$\in$" operator:

Definition 21: For X a full relation and A a per,

$$X \in A \ \ \widehat{=}\ \ X \sqsubseteq A \ \wedge \ X \circ A = X$$

We are now able to make precise the disjointness property of elements:

Lemma 22: $X, Y \in A \ \Rightarrow \ \mathit{disj}.X.Y \ \vee \ X = Y$

Proof

$$
\begin{array}{ll}
 & X, Y \in A \Rightarrow \mathit{disj}.X.Y \vee X = Y \\
\equiv & \quad \{ \text{ shunting } \} \\
 & X, Y \in A \wedge \neg(\mathit{disj}.X.Y) \Rightarrow X = Y \\
\equiv & \quad \{ \text{ def } \mathit{disj}, X \text{ and } Y \text{ are pers } \} \\
 & X, Y \in A \wedge X \circ Y \neq \bot\!\!\bot \Rightarrow X = Y \\
\equiv & \quad \{ \text{ cone axiom } \} \\
 & X, Y \in A \wedge \top\!\!\top \circ X \circ Y \circ \top\!\!\top = \top\!\!\top \Rightarrow X = Y \\
\equiv & \quad \{ \text{ lemma below } \} \\
 & X, Y \in A \wedge \top\!\!\top \circ X \circ Y \circ \top\!\!\top = \top\!\!\top \Rightarrow \top\!\!\top \circ X = \top\!\!\top \circ Y
\end{array}
$$

We proceed now to demonstrate $\top\!\!\top \circ X = \top\!\!\top \circ Y$ by direct calculuation. We only show one inclusion; the other follows similarly.

$$
\begin{array}{ll}
 & \top\!\!\top \circ X \\
= & \quad \{ \ X \in A \ \} \\
 & \top\!\!\top \circ X \circ A \\
\sqsupseteq & \quad \{ \ Y \in A \ \} \\
 & \top\!\!\top \circ X \circ Y \\
= & \quad \{ \ Y \text{ is a full relation } \} \\
 & \top\!\!\top \circ X \circ Y \circ \top\!\!\top \circ Y \\
= & \quad \{ \ \top\!\!\top \circ X \circ Y \circ \top\!\!\top = \top\!\!\top \ \} \\
 & \top\!\!\top \circ Y
\end{array}
$$

Used in the first part of the disjointness proof is the lemma that for full relations X and Y, the equality $X = Y$ is equivalent to $\top\!\!\top \circ X = \top\!\!\top \circ Y$. The "$\Rightarrow$" direction follows by Leibniz law; for the "$\Leftarrow$" direction, we calculate as follows:

$$
\begin{array}{ll}
 & X \\
= & \qquad \{ \ X \text{ is a full relation } \} \\
 & X \circ \top\!\!\top \circ X\cup \\
= & \qquad \{ \ \top\!\!\top \circ X = \top\!\!\top \circ Y \ \} \\
 & Y \circ \top\!\!\top \circ Y\cup \\
= & \qquad \{ \ Y \text{ is a full relation } \} \\
 & Y
\end{array}
$$

The "$\in$" operator is sometimes used in combination with the following:

Axiom 23: (extensionality)

$$per.A \;\;\Rightarrow\;\; A = \bigsqcup (X \;:\; X \in A \;:\; X)$$

By expanding for "$\in$" we see that $A \sqsupseteq \bigsqcup (X : X \in A : X)$ always holds, i.e. pers contain all their elements. The extensionality axiom makes a stronger statement: pers contain nothing but their elements. In combination with the disjointness lemma given earlier, extensionality makes precise the statement that every per can be written in a unique way as the union of disjoint full relations. There are many equivalent formulations of extensionality in the spec calculus [10]. It is important to note that extensionality axioms are independent of the spec–calculi axioms: there are simple models of the spec calculus in which extensionality does not hold.

To clarify our interpretation of pers as types, let us consider an example. A *rational number* may be represented as a pair of integers (a, b), with $b \neq 0$. The interpretation is that (a, b) represents the rational a/b. This representation is not unique: pairs (a, b) and (c, d) represent the same rational iff $ad = bc$. We see then that rational numbers induce a per, RAT say, on the set $\mathbb{Z}\times\mathbb{Z}$:

$$(a, b)\; RAT\; (c, d) \;\;\hat{=}\;\; ad = bc \;\wedge\; bd \neq 0$$

Partiality arises in the denominator of a rational being restricted to non–zero integers; equivalence classes arise in the representation as pairs of integers being non–unique. Viewing the per RAT as a type, elements of RAT are not simply pairs of integers, but full relations on equivalence classes of pairs of integers.

We conclude this section by noting a special property of the per $\top\!\!\top$. Being the full relation on the implicit universe, $\top\!\!\top$ comprises precisely one element, namely $\top\!\!\top$ itself. The importance of such a *unit–type* is well known from category theory. In the monotypes world, no such convenient spec exists already; we are required to postulate the existence of a unit–type by means of an extra axiom [3].

3.1 Orderings on types

Consider the types of (non–empty) *trees*, *lists* and *uplists*. (We use the term *uplist* for a list whose elements are ascending with respect to some partial ordering.) Elements of all three types can be built from constructors $[-]$ (making singleton elements) and "$+\!\!\!+$" (joining two elements together). The three types are different however in the properties required of the constructors. We introduce in this section three orderings on types; these orderings allow us to make precise such relationships.

When used as a constructor for trees, "$+\!\!\!+$" is not subject to any laws. When used with lists however, we require that "$+\!\!\!+$" be associative. For example, we require that $[1] +\!\!\!+ ([2] +\!\!\!+ [3]) = ([1] +\!\!\!+ [2]) +\!\!\!+ [3]$. More formally, while the two expressions form distinct (singleton) equivalence classes in the type of trees, they are the members of the same equivalence class in the type of lists. In standard list notation, this equivalence class is denoted by $[1, 2, 3]$. With uplists the "$+\!\!\!+$" constructor becomes partial, in that certain constructions are forbidden. In particular, $X +\!\!\!+ Y$ is only defined if the last element of X is at most the first element of Y. For example, although $[1, 3]$ and $[2, 4]$ are both uplists, their combination, $[1, 3, 2, 4]$ is not. More formally, $[1, 3, 2, 4]$ does not represent an equivalence class in the type of uplists.

We introduce now three orderings on types: "$\subseteq$", "$\lhd\!|$", and "$\lhd$". Writing *uplist* $\subseteq$ *list* will express that uplists are a special kind of list: every uplist is a list, but the reverse in general does not hold. Writing *list* $\lhd\!|$ *tree* will say that every list is an equivalence class of trees. Writing *uplist* $\lhd$ *tree* will combine the previous two statements: uplists are made from trees by coalescing and discarding equivalence classes. Each of the three operators is first defined using the "$\in$" operator, after which an equivalent but simpler formulation is given.

Definition 24: Given pers A and B,

$$A \subseteq B \;\; \hat{=} \;\; \forall\,(X : X \in A : X \in B)$$

We read $A \subseteq B$ as "A is a subtype of B", i.e. the type A is formed by discarding elements of type B. We observe now that "$\subseteq$" is a partial ordering on pers, with $\perp\!\!\perp$ as a least element. (Pers in fact form a "meet semi–lattice" under "$\subseteq$".)

Lemma 25: $A \subseteq B \;\; \equiv \;\; A \circ B = A \;\wedge\; A \sqsubseteq B$

Proof "$\Leftarrow$": expanding for $X \in B$ gives two parts

$$\begin{array}{ll} & B \\ \sqsupseteq & \quad \{ \;\; B \sqsupseteq A \;\; \} \\ & A \\ \sqsupseteq & \quad \{ \;\; X \in A \;\; \} \\ & X \end{array}$$

$$
\begin{array}{ll}
 & X \circ B \\
= & \quad \{ \ X \in A \ \} \\
 & X \circ A \circ B \\
= & \quad \{ \ A \circ B = A \ \} \\
 & X \circ A \\
= & \quad \{ \ X \in A \ \} \\
 & X
\end{array}
$$

Proof "$\Rightarrow$"

$$
\begin{array}{ll}
 & A \circ B \\
= & \quad \{ \ \text{extensionality} \ \} \\
 & A \circ \bigsqcup(X : X \in B : X) \\
= & \quad \{ \ \text{distribution} \ \} \\
 & \bigsqcup(X : X \in B : A \circ X) \\
= & \quad \{ \ \text{range–splitting} \ \} \\
 & \bigsqcup(X : X \in B \wedge X \notin A : A \circ X) \sqcup \bigsqcup(X : X \in B \wedge X \in A : A \circ X) \\
= & \quad \{ \ \forall (X : \ X \in A \ : \ X \in B) \ \} \\
 & \bigsqcup(X : X \in B \wedge X \notin A \ : \ A \circ X) \sqcup \bigsqcup(X : X \in A : A \circ X) \\
= & \quad \{ \ \text{def} \in \ \} \\
 & \bigsqcup(X : X \in B \wedge X \notin A : A \circ X) \sqcup \bigsqcup(X : X \in A : X) \\
= & \quad \{ \ \text{extensionality} \ \} \\
 & \bigsqcup(X : X \in B \wedge X \notin A : \bigsqcup(Y : Y \in A : Y) \circ X) \sqcup A \\
= & \quad \{ \ \text{distribution} \ \} \\
 & \bigsqcup(X,Y : X \in B \wedge X \notin A \wedge Y \in A : Y \circ X) \sqcup A \\
= & \quad \{ \ \forall (Y : \ Y \in A \ : \ Y \in B), X \notin A \ \wedge \ Y \in A \Rightarrow X \neq Y \ \} \\
 & \bigsqcup(X,Y : X,Y \in B \wedge X \notin A \wedge Y \in A \wedge X \neq Y : Y \circ X) \sqcup A \\
= & \quad \{ \ \text{disjointness: } X,Y \in B \ \wedge \ X \neq Y \ \Rightarrow \ Y \circ X = \perp\!\!\perp \ \} \\
 & A
\end{array}
$$

$$
\begin{array}{ll}
 & B \\
= & \quad \{ \ \text{extensionality} \ \} \\
 & \bigsqcup(X : X \in B : X) \\
\sqsupseteq & \quad \{ \ \forall (X : \ X \in A \ : \ X \in B) \ \} \\
 & \bigsqcup(X : X \in A : X) \\
= & \quad \{ \ \text{extensionality} \ \} \\
 & A
\end{array}
$$

Given a full relation X and a per A, we say that X *coalesces* A if all the elements of A are combined to form the single element of X. For example, $\{a,b,c\}^2$ coalesces $\{a,b\}^2 \cup \{c\}^2$. In the spec calculus, coalescence can be defined as follows:

Definition 26: For X a full relation and A a per,

$$X \ \mathit{coalesces} \ A \ \ \widehat{=} \ \ X = A \circ \top\!\!\top \circ A$$

This notion is used in the definition of the "$\lhd$" ordering:

Definition 27: Given pers A and B,

$$A \lhd B \ \ \hat{=}\ \ \forall\,(X : X \in A : \exists\,(C : C \subseteq B : X \ \mathit{coalesces}\ C))$$

We read $A \lhd B$ as "A is a per on B" — the type A is formed by discarding and coalescing elements of type B. We note that "$\lhd$" is a partial ordering on pers, with "$\perp\!\!\perp$" as least element, and "I" as greatest element. (Pers in fact form a lattice under "$\lhd$".) We work in practice with a simpler formulation:

Lemma 28: $A \lhd B \ \equiv\ A \circ B = A$

Proof "$\Leftarrow$"

$$
\begin{array}{ll}
 & \exists\,(C : C \subseteq B : X = C \circ \top\!\!\top \circ C) \\
\Leftarrow & \qquad \{\ C := X{>} \circ B\ \} \\
 & X{>} \circ B \subseteq B \ \wedge\ X = X{>} \circ B \circ \top\!\!\top \circ X{>} \circ B \\
\Leftarrow & \qquad \{\ \text{lemma } 89\ \} \\
 & X \circ B = X \ \wedge\ X = X{>} \circ B \circ \top\!\!\top \circ B \circ X{>} \\
\equiv & \qquad \{\ X \in A, \text{domains}, \mathit{per}.B\ \} \\
 & X \circ A \circ B = X \ \wedge\ X = X{>} \circ B{>} \circ \top\!\!\top \circ B{>} \circ X{>} \\
\equiv & \qquad \{\ A \circ B = A,\ X{>} \sqsubseteq \{X \in A\}\ A{>} \sqsubseteq \{A \circ B = A\}\ B{>}\ \} \\
 & X \circ A = X \ \wedge\ X = X{>} \circ \top\!\!\top \circ X{>} \\
\equiv & \qquad \{\ X \in A, X \text{ is a full relation}\ \} \\
 & \mathit{true}
\end{array}
$$

Proof "$\Rightarrow$"

$$
\begin{array}{ll}
 & A \circ B \\
= & \qquad \{\ \text{extensionality}\ \} \\
 & \bigsqcup\,(X : X \in A : X) \circ B \\
= & \qquad \{\ \text{distribution}\ \} \\
 & \bigsqcup\,(X : X \in A : X \circ B) \\
= & \qquad \{\ \forall\,(X : X \in A : \exists\,(C : C \subseteq B : X = C \circ \top\!\!\top \circ C))\ \} \\
 & \bigsqcup\,(X, C : X \in A \wedge C \subseteq B \wedge X = C \circ \top\!\!\top \circ C : X \circ B) \\
= & \qquad \{\ \text{substitution}\ \} \\
 & \bigsqcup\,(X, C : X \in A \wedge C \subseteq B \wedge X = C \circ \top\!\!\top \circ C : C \circ \top\!\!\top \circ C \circ B) \\
= & \qquad \{\ C \subseteq B\ \} \\
 & \bigsqcup\,(X, C : X \in A \wedge C \subseteq B \wedge X = C \circ \top\!\!\top \circ C : C \circ \top\!\!\top \circ C) \\
= & \qquad \{\ \text{substitution}\ \} \\
 & \bigsqcup\,(X, C : X \in A \wedge C \subseteq B \wedge X = C \circ \top\!\!\top \circ C : X) \\
= & \qquad \{\ \forall\,(X : X \in A : \exists\,(C : C \subseteq B : X = C \circ \top\!\!\top \circ C))\ \} \\
 & \bigsqcup\,(X : X \in A : X) \\
= & \qquad \{\ \text{extensionality}\ \} \\
 & A
\end{array}
$$

From the symmetry of A and B (i.e. $A = A^{\cup}$ and $B = B^{\cup}$), it follows that $A \lhd B \ \equiv \ B \circ A = A$. We note also that "$\subseteq$" can be expressed in terms of "$\lhd$":

Lemma 29: $A \subseteq B \ \equiv \ A \lhd B \ \wedge \ A \sqsubseteq B$

Writing $A \lhd B$ expresses that A is a partial equivalence relation on B. If A is an equivalence relation on B (i.e. no elements have been discarded in moving to A from B) we write $A \mathrel{\lhd\!|} B$. It can be shown that "$\lhd\!|$" is a partial ordering on types, with $\top\!\!\top$ as greatest element. (Pers form a "join semi–lattice" under "$\lhd\!|$".)

Definition 30: Given pers A and B,

$$A \mathrel{\lhd\!|} B \ \ \widehat{=} \ \ A \lhd B \ \wedge \ \forall\,(X : X \in B : \exists\,(Y : Y \in A : Y \sqsupseteq X))$$

Lemma 31: $A \mathrel{\lhd\!|} B \ \equiv \ A \lhd B \wedge A \sqsupseteq B$

Proof "$\Rightarrow$"

$$\begin{array}{ll}
 & B \\
= & \quad \{ \ \text{extensionality} \ \} \\
 & \bigsqcup\,(X : X \in B : X) \\
= & \quad \{ \ \forall\,(X : X \in B : \exists\,(Y : Y \in A : Y \sqsupseteq X)) \ \} \\
 & \bigsqcup\,(X, Y : X \in B \wedge Y \in A \wedge Y \sqsupseteq X : X) \\
\sqsubseteq & \quad \{ \ Y \sqsupseteq X \ \} \\
 & \bigsqcup\,(X, Y : X \in B \wedge Y \in A \wedge Y \sqsupseteq X : Y) \\
\sqsubseteq & \quad \{ \ Y \in A \ \} \\
 & \bigsqcup\,(X, Y : X \in B \wedge Y \in A \wedge Y \sqsupseteq X : A) \\
\sqsubseteq & \quad \{ \ \text{calculus} \ \} \\
 & A
\end{array}$$

Proof "$\Leftarrow$"

$$\begin{array}{ll}
 & \exists\,(Y : Y \in A : Y \sqsupseteq X) \\
\Leftarrow & \quad \{ \ Y := A \circ X \circ A \ \} \\
 & A \circ X \circ A \in A \ \wedge \ A \circ X \circ A \sqsupseteq X \\
\equiv & \quad \{ \ \text{def} \in \ \} \\
 & A \circ X \circ A = A \circ X \circ A \circ \top\!\!\top \circ (A \circ X \circ A)^{\cup} \ \wedge \ A \sqsupseteq A \circ X \circ A \ \wedge \\
 & A \circ X \circ A \circ A = A \circ X \circ A \ \wedge \ A \circ X \circ A \sqsupseteq X \\
\Leftarrow & \quad \{ \ A \text{ and } X \text{ are pers, } A \sqsupseteq B,\ X \in B \ \} \\
 & A \circ X \circ A = A \circ X \circ A \circ \top\!\!\top \circ A \circ X \circ A \ \wedge \ A \sqsupseteq A \circ B \circ A \ \wedge \ B \circ X \circ B \sqsupseteq X \\
\equiv & \quad \{ \ A \lhd B,\ X \in B,\ X \text{ is a full relation} \ \} \\
 & A \circ X \circ \top\!\!\top \circ X \circ A = A \circ X \circ A \circ \top\!\!\top \circ A \circ X \circ A \\
\equiv & \quad \{ \ \top\!\!\top \sqsupseteq A \circ \top\!\!\top \circ A \ \} \\
 & A \circ X \circ A \circ \top\!\!\top \circ A \circ X \circ A \sqsupseteq A \circ X \circ \top\!\!\top \circ X \circ A \\
\equiv & \quad \{ \ X \text{ is a full relation} \ \}
\end{array}$$

$$
\begin{array}{ll}
& A \circ X \circ A \circ \top\!\top \circ A \circ X \circ A \sqsupseteq A \circ X \circ A \\
\Leftarrow & \quad \{ \text{ calculus } \} \\
& A \circ X \circ A \circ (A \circ X \circ A)^{\cup} \circ A \circ X \circ A \sqsupseteq A \circ X \circ A \\
\equiv & \quad \{ \text{ triple rule } \} \\
& true
\end{array}
$$

For reference, let us summarise the three orderings on pers:

$$
\begin{array}{llll}
A \lhd B & \equiv & A \circ B = A & \text{(per ordering)} \\
A \subseteq B & \equiv & A \lhd B \;\wedge\; A \sqsubseteq B & \text{(subtype ordering)} \\
A \lhd\!\mid B & \equiv & A \lhd B \;\wedge\; A \sqsupseteq B & \text{(equiv ordering)}
\end{array}
$$

3.2 Typing judgements for specs

In set–theoretic approaches to relational calculus, it is common to write $R \subseteq A \times B$ (where A and B are sets) in the form of a *typing judgement* $R \in A \sim B$. In this section we seek to define $A \sim B$ with A and B as pers rather than sets.

In writing $R \in A \sim B$, we require that R on its left–side respects the elements of A, and on its right–side respects the elements of B. Let us consider the A part in more detail. Recall that elements of pers are full relations. By a relation R "respecting on its left–side" a full relation $\mathcal{S}^2$, we mean that the image through R of each member of $\mathcal{S}$ is the same set; that is, we require that

$$\forall\,(a,b,c : a,b \in \mathcal{S} : a\ R\ c \;\equiv\; b\ R\ c)$$

(That such an image set may be empty means that R is not required to be total on A.) We call such a full relation $\mathcal{S}^2$ a *left equivalence class* of R. We find that the predicate above can be written in relational calculus as $R \sqsupseteq \mathcal{S}^2 \circ R$:

$$
\begin{array}{ll}
& \mathcal{S}^2 \circ R \sqsubseteq R \\
\equiv & \quad \{ \text{ def } \sqsubseteq \ \} \\
& \forall\,(a,b \;:\; a\ \mathcal{S}^2 \circ R\ b \;:\; a\ R\ b) \\
\equiv & \quad \{ \text{ def } \circ \ \} \\
& \forall\,(a,b \;:\; \exists\,(c :: a\ \mathcal{S}^2\ c \wedge c\ R\ b) \;:\; a\ R\ b) \\
\equiv & \quad \{ \ \exists \text{ elimination } \} \\
& \forall\,(a,b,c \;:\; a\ \mathcal{S}^2\ c \wedge c\ R\ b \;:\; a\ R\ b) \\
\equiv & \quad \{ \text{ def } \mathcal{S}^2 \ \} \\
& \forall\,(a,b,c \;:\; a,c \in S \wedge c\ R\ b \;:\; a\ R\ b) \\
\equiv & \quad \{ \text{ shunting } \} \\
& \forall\,(a,b,c \;:\; a,c \in S : \; c\ R\ b \Rightarrow a\ R\ b) \\
\equiv & \quad \{ \text{ predicate calculus } \} \\
& \forall\,(a,b,c \;:\; a,c \in S : \; a\ R\ b \equiv c\ R\ b)
\end{array}
$$

Similarly we can define a *right equivalence class* of a relation R as a full relation $\mathcal{S}^2$ for which $R \sqsupseteq R \circ \mathcal{S}^2$. We make now two definitions in the spec calculus:

Definition 32:

A *left equivalence class* of a spec R is a full relation X for which $R \sqsupseteq X \circ R$;

A *right equivalence class* of R is a full relation X for which $R \sqsupseteq R \circ X$.

Using these notions we can define the "~" operator:

Definition 33: For A and B pers, $R \in A\sim B \quad \widehat{=}$

$$\begin{array}{l} \forall\,(X \ : \ X \in A \ : \ R \sqsupseteq X \circ R) \ \wedge \ A \circ R \sqsupseteq R \ \wedge \\ \forall\,(X \ : \ X \in B \ : \ R \sqsupseteq R \circ X) \ \wedge \ R \circ B \sqsupseteq R \end{array}$$

We call such a per A a *left domain* of R, and B a *right domain*. The clauses $A \circ R \sqsupseteq R$ and $R \circ B \sqsupseteq R$ above, equivalent respectively to $A \sqsupseteq R<$ and $B \sqsupseteq R>$, are needed to ensure that pers A and B are big enough to serve as domains of R. In practice we work with a simpler but equivalent formulation of "~":

Lemma 34: $R \in A\sim B \quad \equiv \quad A \circ R = R = R \circ B$

Proof (we show only the A part; that for B proceeds similarly)

$$\begin{array}{ll} & \forall\,(X : \ X \in A : \ R \sqsupseteq X \circ R) \ \wedge \ A \circ R \sqsupseteq R \\ \equiv & \qquad \{ \ \text{calculus} \ \} \\ & R \sqsupseteq \bigsqcup(X \ : \ X \in A \ : \ X \circ R) \ \wedge \ A \circ R \sqsupseteq R \\ \equiv & \qquad \{ \ \text{distribution} \ \} \\ & R \sqsupseteq \bigsqcup(X \ : \ X \in A \ : \ X) \circ R \ \wedge \ A \circ R \sqsupseteq R \\ \equiv & \qquad \{ \ \text{extensionality} \ \} \\ & R \sqsupseteq A \circ R \ \wedge \ A \circ R \sqsupseteq R \\ \equiv & \qquad \{ \ \text{calculus} \ \} \\ & A \circ R = R \end{array}$$

The right–side of Lemma 34 can be written in fact as a single equality:

Lemma 35: $R \in A\sim B \quad \equiv \quad A \circ R \circ B = R$

Proof "$\Rightarrow$"

$$\begin{array}{ll} & A \circ R \circ B \\ = & \qquad \{ \ A \circ R = R \ \} \\ & R \circ B \\ = & \qquad \{ \ R \circ B = R \ \} \\ & R \end{array}$$

Proof "$\Leftarrow$" (We prove only one conjunct; the other proceeds similarly)

$$\begin{array}{ll}
& A \circ R \\
= & \quad \{ \ A \circ R \circ B = R \ \} \\
& A \circ A \circ R \circ B \\
= & \quad \{ \ per.A \ \} \\
& A \circ R \circ B \\
= & \quad \{ \ \text{assumption again} \ \} \\
& R
\end{array}$$

We conclude with some properties of the "$\sim$" operator:

- $R \in A{\sim}B \quad \Leftarrow \quad R \in C{\sim}D \ \wedge \ C \lhd A \ \wedge \ D \lhd B$
- $R \circ S \in A{\sim}D \quad \Leftarrow \quad R \in A{\sim}B \ \wedge \ S \in C{\sim}D$
- $R{\cup} \in B{\sim}A \ \equiv \ R \in A{\sim}B$

The first law above tells us that from a valid typing of R, we may break apart or add new elements to the left and right domains and still have a valid typing; an immediate consequence is that a spec need not have a unique typing under "$\sim$". (For a general discussion of "least" and "greatest" typings in the pers as types world, the reader is referred to Voermans article [11].) The second law gives the link between "$\sim$" and composition; we may have expected some relationship between the intermediate pers B and C, but this in fact is not required. The third and final law above tells us that "$\sim$" behaves as we would expect with respect to the "$\cup$" operator.

3.3 Typing judgements for difunctionals

We view relational programming as a generalisation of functional programming: relations for us are the primitive notion, functions are regarded as special kinds of relation. In the last section we introduced a typing notation $R \in A{\sim}B$ for specs. In this section we introduce a typing notation $f \in A \leftarrow B$ for functional specs.

Normally the "input part" of a functional relation is on its left–side. Following [3] however, we view the right–side of a functional relation as its input part, writing $A \leftarrow B$ for the set of all functions to A from B. This choice avoids confusion between written and diagrammatic order for composition of functions, and is consistent with placing the argument to the right of the function symbol during application.

> **Definition 36:** Give a spec R and a full relation X, the *image* of X right–left through R is given by the full relation $R.X \ \hat{=} \ R \circ X \circ R{\cup}$

In writing $f \in A \leftarrow B$, we require two properties: f has a valid typing $A{\sim}B$ as a spec (i.e. typed functions are a special case of typed specs); applying f to an element of per B gives an element of per A (i.e. f is functional to A from B.)

Definition 37:

$$f \in A \leftarrow B \;\; \hat{=} \;\; f \in A{\sim}B \;\; \wedge \;\; \forall\,(X \; : \; X \in B \; : \; f.X \in A)$$

In practice as usual we work with a simpler but equivalent formulation:

Lemma 38: $f \in A \leftarrow B \;\; \equiv \;\; f \in A{\sim}B \;\; \wedge \;\; A \sqsupseteq f \circ f{\cup}$

Proof "$\Rightarrow$"

$$\begin{array}{ll}
 & A \\
\sqsupseteq & \qquad \{ \text{ assumption } \} \\
 & \bigsqcup(X \; : \; X \in B \; : \; f \circ X \circ f{\cup}) \\
= & \qquad \{ \text{ distribution } \} \\
 & f \circ \bigsqcup(X \; : \; X \in B \; : \; X) \circ f{\cup} \\
= & \qquad \{ \text{ extensionality } \} \\
 & f \circ B \circ f{\cup} \\
= & \qquad \{ \; f \circ B = f \; \} \\
 & f \circ f{\cup}
\end{array}$$

Proof "$\Leftarrow$"

$$\begin{array}{ll}
 & f.X \in A \\
\equiv & \qquad \{ \text{ def } . \; \} \\
 & f \circ X \circ f{\cup} \in \; A \\
\equiv & \qquad \{ \text{ def } \in \; \} \\
 & A \sqsupseteq \; f \circ X \circ f{\cup} \;\; \wedge \;\; A \circ f \circ X \circ f{\cup} \; = \; f \circ X \circ f{\cup} \\
\equiv & \qquad \{ \; A \circ f = f \; \} \\
 & A \sqsupseteq \; f \circ X \circ f{\cup} \\
\Leftarrow & \qquad \{ \; X \in B \; \} \\
 & A \sqsupseteq \; f \circ B \circ f{\cup} \\
\equiv & \qquad \{ \; f \circ B = f, \;\; A \sqsupseteq f \circ f{\cup} \; \} \\
 & true
\end{array}$$

Sometimes we are not interested in the right–domain of f:

Definition 39: f is *functional to* $A \;\; \hat{=} \;\; A \circ f = f \;\; \wedge \;\; A \sqsupseteq f \circ f{\cup}$

In the monotypes world [3], functional specs are characterised as those f satisfying $I \sqsupseteq f \circ f{\cup}$. It is shown below how this definition corresponds to the normal set–theoretic definition of a relation being functional:

$$\begin{array}{ll}
 & I \sqsupseteq \; f \circ f{\cup} \\
\equiv & \qquad \{ \text{ def } \sqsupseteq \; \} \\
 & \forall\,(x, y :: \; x \; I \; y \Leftarrow x \; f \circ f{\cup} \; y) \\
\equiv & \qquad \{ \text{ def } I, \circ \; \}
\end{array}$$

$$\begin{array}{ll} & \forall(x,y::\ x=y \Leftarrow \exists(z::x\ f\ z \wedge z\ f^{\cup}\ y)) \\ \equiv & \qquad \{\ \text{def}\ {}^{\cup}\ \} \\ & \forall(x,y::\ x=y \Leftarrow \exists(z::x\ f\ z \wedge y\ f\ z)) \\ \equiv & \qquad \{\ \exists\ \text{elimination}\ \} \\ & \forall(x,y,z::\ x=y\ \Leftarrow\ x\ f\ z \wedge y\ f\ z) \end{array}$$

We see then that $I \sqsupseteq f \circ f^{\cup}$ expresses that f is a partial function: for all x, there exists at most one y such that $y\ f\ x$. In keeping with [3], we use the term *imp* (abbreviating "implementation") for a spec f (recall, spec abbreviates "specification") satisfying $I \sqsupseteq f \circ f^{\cup}$. Using the special term "imp" rather than simply "function" avoids confusion with the other notion of functionality introduced in this paper, that in the pers as types setting (for which we use the special term "difunctional".)

Definition 40: $imp.f \ \ \widehat{=}\ \ I \sqsupseteq f \circ f^{\cup}$

If $f^{\cup}$ is a imp, we refer to f itself as a *co–imp*; just as imps correspond to functional relations, so co–imps correspond to injective relations. (As noted by van Gasteren in [7], the symmetry between functional and injective relations is lost in many texts, through restricting the notion of injectivity to functions.)

Definition 41: $co\text{–}imp.f \ \ \widehat{=}\ \ imp.f^{\cup}$

The notion of functionality is not case in stone, but depends upon our notion of type. Working with monotypes, we have $imp.f \ \widehat{=}\ I \sqsupseteq f \circ f^{\cup}$. We assert that with pers as types, the functional specs are precisely those satisfying $f \sqsupseteq f \circ f^{\cup} \circ f$. Such specs are known as *difunctionals* [12]. (Difunctional relations have also been called "regular" relations and "pseudo–invertible" relations [13].)

Definition 42: $difun.f \ \ \widehat{=}\ \ f \ \sqsupseteq\ f \circ f^{\cup} \circ f$

We always use small letters $f, g, h, \ldots$ to denote difunctional specs. (This does not clash with the use of these letters to denote imps, since as we shall see, every imp is difunctional.) Since the triple rule $R \sqsubseteq R \circ R^{\cup} \circ R$ holds for all specs R, we are free to replace the containment in the above definition by an equality.

Lemma 43: $difun.f \ \ \equiv\ \ f = f \circ f^{\cup} \circ f$

Our assertion that the functional specs are precisely the difunctionals is verified by the three results below. The first shows that every functionally typed spec is difunctional, the remaining two that every difunctional can be functionally typed.

Lemma 44: $f \in A \leftarrow B \ \ \Rightarrow\ \ difun.f$

Lemma 45: $difun.f \ \ \Rightarrow\ \ per.(f \circ f^{\cup})\ \wedge\ per.(f^{\cup} \circ f)$

Lemma 46: $difun.f \;\Rightarrow\; f \in f \circ f^{\cup} \leftarrow f^{\cup} \circ f$

Proof 44

$$
\begin{array}{ll}
 & f \\
= & \qquad \{ \; A \circ f = f \; \} \\
 & A \circ f \\
\sqsupseteq & \qquad \{ \; A \sqsupseteq f \circ f^{\cup} \; \} \\
 & f \;\circ\; f^{\cup} \;\circ\; f
\end{array}
$$

Expanding for "$\leftarrow$" in Lemma 46 gives precisely the assumption that f be difunctional. We are however required to show that under this assumption, $f \circ f^{\cup}$ and $f^{\cup} \circ f$ are types, as in Lemma 45. We prove only the first conjunct of this result. Symmetry of $f \circ f^{\cup}$ is trivial; for transitivity, we calculate as follows:

$$
\begin{array}{ll}
 & transitive.(f \circ f^{\cup}) \\
\equiv & \qquad \{ \; \text{def} \; \} \\
 & f \circ f^{\cup} \sqsupseteq f \circ f^{\cup} \circ f \circ f^{\cup} \\
\Leftarrow & \qquad \{ \; \text{monotonicity} \; \} \\
 & f \sqsupseteq f \circ f^{\cup} \circ f \\
\Leftarrow & \qquad \{ \; \text{def} \; \} \\
 & difun.f
\end{array}
$$

Lemma 46 above encodes that the pers $f \circ f^{\cup}$ and $f^{\cup} \circ f$ are respectively left and right domains for difunctional f. We conclude this section with the result that these pers are in fact the least domains for f under the "$\lhd$" ordering.

Lemma 47: $difun.f \;\Rightarrow\; (f \in A \sim B \;\equiv\; f \circ f^{\cup} \lhd A \;\wedge\; f^{\cup} \circ f \lhd B)$

Proof "$\Rightarrow$"

$$
\begin{array}{ll}
 & f \circ f^{\cup} \lhd A \;\wedge\; f^{\cup} \circ f \lhd B \\
\equiv & \qquad \{ \; \text{def} \lhd \; \} \\
 & f \circ f^{\cup} \circ A = f \circ f^{\cup} \;\wedge\; f^{\cup} \circ f \circ B = f^{\cup} \circ f \\
\equiv & \qquad \{ \; \text{assumption: } f \circ B = f \; \} \\
 & f \circ f^{\cup} \circ A = f \circ f^{\cup} \\
\equiv & \qquad \{ \; {}^{\cup} \text{ both sides, } per.\text{A} \; \} \\
 & A \circ f \circ f^{\cup} = f \circ f^{\cup} \\
\equiv & \qquad \{ \; \text{assumption: } A \circ f = f \; \} \\
 & true
\end{array}
$$

3.4 More about difunctionals

In the last section we showed that with pers as types, the functional specs are precisely the difunctionals. In this section we document some properties of difunctionals. In particular, we give some conditions under which certain operators preserve difunctionality, and give three other ways to think about difunctionals.

Lemma 48: $per.A \;\Rightarrow\; difun.A$

Proof

$$
\begin{array}{ll}
 & A \\
\sqsupseteq & \qquad \{\; per.A \;\Rightarrow\; transitive.A \;\} \\
 & A \circ A \\
= & \qquad \{\; per.A \;\equiv\; A = A \circ A^{\cup} \;\} \\
 & A \circ A^{\cup} \circ A
\end{array}
$$

Lemma 49: $imp.f \;\Rightarrow\; difun.f$

Proof

$$
\begin{array}{ll}
 & f \\
= & \qquad \{\; \text{identity} \;\} \\
 & I \circ f \\
\sqsupseteq & \qquad \{\; \text{assumption: } I \sqsupseteq f \circ f^{\cup} \;\} \\
 & f \circ f^{\cup} \circ f
\end{array}
$$

Lemma 50: $difun.f \;\equiv\; difun.f^{\cup}$

Proof

$$
\begin{array}{ll}
 & difun.f^{\cup} \\
\equiv & \qquad \{\; \text{def} \;\} \\
 & f^{\cup} \circ f^{\cup\cup} \circ f^{\cup} \;=\; f^{\cup} \\
\equiv & \qquad \{\; \text{distribution} \;\} \\
 & (f \circ f^{\cup} \circ f)^{\cup} = f^{\cup} \\
\equiv & \qquad \{\; \text{reverse} \;\} \\
 & f \circ f^{\cup} \circ f \;=\; f \\
\equiv & \qquad \{\; \text{def} \;\} \\
 & difun.f
\end{array}
$$

The above law is where the prefix "di" in difunctional comes from, telling us that every difunctional is not just functional, but in fact *bijective*. In conjunction with the previous lemma, we see then that the every co–imp is difunctional.

Difunctionals are closed under intersection:

Lemma 51: $\mathit{difun}.(f \sqcap g) \;\;\Leftarrow\;\; \mathit{difun}.f \;\wedge\; \mathit{difun}.g$

Proof

$$
\begin{array}{ll}
 & f \sqcap g \\
= & \qquad \{\; f{,}g \text{ difunctional } \} \\
 & f \circ f^{\cup} \circ f \;\sqcap\; g \circ g^{\cup} \circ g \\
\sqsupseteq & \qquad \{\; \text{monotonicity of} \circ \text{and} \sqcap \; \} \\
 & (f \sqcap g) \;\circ\; (f^{\cup} \sqcap g^{\cup}) \;\circ\; (f \sqcap g) \\
= & \qquad \{\; \text{reverse} \; \} \\
 & (f \sqcap g) \;\circ\; (f \sqcap g)^{\cup} \;\circ\; (f \sqcap g)
\end{array}
$$

We might suspect difunctionals to be similarly preserved under "$\sqcup$", but this is not so. Disjointness provides however a simple condition to ensure preservation:

Lemma 52: $\mathit{difun}.(f \sqcup g) \;\;\Leftarrow\;\; \mathit{difun}.f \;\wedge\; \mathit{difun}.g \;\wedge\; \mathit{disj}.f.g$

Proof

$$
\begin{array}{ll}
 & \mathit{difun}.(f \sqcup g) \\
\equiv & \qquad \{\; \text{def, reverse} \; \} \\
 & f \sqcup g \;\sqsupseteq\; (f \sqcup g) \;\circ\; (f^{\cup} \sqcup g^{\cup}) \;\circ\; (f \sqcup g) \\
\equiv & \qquad \{\; \text{distribution, lemma 16} \; \} \\
 & f \sqcup g \;\sqsupseteq\; f \circ f^{\cup} \circ f \;\sqcup\; g \circ g^{\cup} \circ g \\
\Leftarrow & \qquad \{\; \text{calculus} \; \} \\
 & f \;\sqsupseteq\; f \circ f^{\cup} \circ f \;\;\wedge\;\; g \;\sqsupseteq\; g \circ g^{\cup} \circ g \\
\equiv & \qquad \{\; \text{def difun} \; \} \\
 & \mathit{difun}.f \wedge \mathit{difun}.g
\end{array}
$$

In general, difunctionals are not closed under composition. For example,

$$f = \{(a,x),(a,y),(b,z)\}$$

$$g = \{(x,a),(y,b),(z,b)\}$$

are both difunctional (f is an imp, g is a co–imp), but their composition $f \circ g$ is not, as the reader may wish to verify. A simple "type check" however is all that is needed to ensure closure: the composition $(f \circ g)$ of two difunctionals is itself difunctional provided that the least right domain of f subsumes the least left domain of g:

Lemma 53: $difun.(f \circ g) \;\Leftarrow\; difun.f \;\wedge\; difun.g \;\wedge\; f^{\cup} \circ f \;\sqsupseteq\; g \circ g^{\cup}$

Proof

$$
\begin{array}{ll}
 & difun.(f \circ g) \\
\equiv & \quad \{ \text{ def } \} \\
 & f \circ g \;\sqsupseteq\; f \circ g \circ g^{\cup} \circ f^{\cup} \circ f \circ g \\
\Leftarrow & \quad \{ \text{ assumption: } f^{\cup} \circ f \;\sqsupseteq\; g \circ g^{\cup} \; \} \\
 & f \circ g \;\sqsupseteq\; f \circ f^{\cup} \circ f \circ f^{\cup} \circ f \circ g \\
\equiv & \quad \{ \text{ assumption: } difun.f \; \} \\
 & f \circ g \sqsupseteq f \circ g \\
\equiv & \quad \{ \text{ calculus } \} \\
 & true
\end{array}
$$

Flipping the role of least right and left domains also works:

Lemma 54: $difun.(f \circ g) \;\Leftarrow\; difun.f \;\wedge\; difun.g \;\wedge\; g \circ g^{\cup} \;\sqsupseteq\; f^{\cup} \circ f$

The proof is similar to that above.

Disjoint bundles

The full relation on a set $\mathcal{S}$ is given by $\mathcal{S} \times \mathcal{S}$. Relaxing the constraint that both sides of a full relation must be the same set gives what we call *bundles*: relations of the form $\mathcal{S} \times \mathcal{T}$, where sets $\mathcal{S}$ and $\mathcal{T}$ may differ. Just as pers can be written in terms of full relations, so difunctionals can be written in terms of bundles:

Lemma 55: every difunctional can be written in a unique way as the union of disjoint bundles.

The bundles interpretation of difunctionals is particularly useful in understanding conditions under which operators such as composition preserve difunctionality.

Co–imp/imp factorisation

Every relation can be factorised in the form $f \circ g^{\cup}$, where f and g are imps. Flipping things around, it is a simple exercise to show that for imps f and g, the spec $f^{\cup} \circ g$ is always difunctional. In fact, we have the following result:

Lemma 56: precisely the difunctional specs can be factorised as the composition of a co–imp and an imp.

This property of difunctionals has important applications in deriving programs that take the form of "representation changers". (See [20] for more details.)

Invertible specs

An *inverse* of a spec $R \in A \sim B$ is a spec $S \in B \sim A$ satisfying $R \circ S = A$ and $S \circ R = B$. Not all specs $R \in A \sim B$ are invertible, but those that are have a unique inverse, namely $R^\cup$. (See Lemma D22 in [3] for the proof.)

Lemma 57: $S \in B \sim A$ is an inverse of $R \in A \sim B \;\Rightarrow\; S = R^\cup$

Which specs can be inverted ? Precisely the difunctionals:

Lemma 58: $R \in A \sim B$ is invertible $\;\Rightarrow\; \mathit{difun}.R$

Lemma 59: $\mathit{difun}.R \;\Rightarrow\; R \in R \circ R^\cup \sim R^\cup \circ R$ is invertible

(That the typing is valid in the last result follows from Lemma 46.)

3.5 More about typing judgements

In writing $f \in A \leftarrow B$, it is not required that f be total on elements of B; it is acceptable that for some $X \in B$, $f.X = \perp\!\!\!\perp$. The notion of a spec being *total* on some type can be defined quite separately from the "$\leftarrow$" operator:

Definition 60: spec R is *total on* per $B \;\hat{=}$

$$R \circ B = R \;\;\wedge\;\; \forall\,(X \;:\; X \in B \wedge X \neq \perp\!\!\!\perp \;:\; R.X \neq \perp\!\!\!\perp)$$

The second conjunct can be written more simply:

Lemma 61: R is *total on* $B \;\equiv\; R \circ B = R \;\;\wedge\;\; R^\cup \circ R \sqsupseteq B$

With respect to the assumption $R \circ B = R$, the conjunct $R^\cup \circ R \sqsupseteq B$ above is equivalent to $R{>} \sqsupseteq B{>}$. It is with this substitution that we prove the "$\Leftarrow$" direction.

Proof "$\Leftarrow$"

$$\begin{array}{ll}
 & R.X \neq \perp\!\!\!\perp \\
\equiv & \quad \{\ \text{def “.”}\ \} \\
 & R \circ X \circ R^\cup \neq \perp\!\!\!\perp \\
\equiv & \quad \{\ \text{cone axiom}\ \} \\
 & \top\!\!\top \circ R \circ X \circ R^\cup \circ \top\!\!\top \;=\; \top\!\!\top \\
\equiv & \quad \{\ R{>} \sqsupseteq B{>},\ \text{domains}\ \} \\
 & \top\!\!\top \circ B \circ X \circ B^\cup \circ \top\!\!\top \;=\; \top\!\!\top \\
\equiv & \quad \{\ X \in B\ \} \\
 & \top\!\!\top \circ X \circ \top\!\!\top = \top\!\!\top \\
\equiv & \quad \{\ X \neq \perp\!\!\!\perp\ \} \\
 & \mathit{true}
\end{array}$$

Proof "$\Rightarrow$"

$$
\begin{array}{ll}
 & R^{\cup} \circ R \\
= & \qquad \{ \;\; R \circ B = R \;\; \} \\
 & B \circ R^{\cup} \circ R \circ B \\
\sqsupseteq & \qquad \{ \;\; \text{def "}\in\text{"} \;\; \} \\
 & \bigsqcup(X \;:\; X \in B \;:\; X \circ R^{\cup} \circ R \circ X) \\
= & \qquad \{ \;\; X \in B \;\; \} \\
 & \bigsqcup(X \;:\; X \in B \;:\; X \circ \top\!\!\top \circ X^{\cup} \circ R^{\cup} \circ R \circ X \circ \top\!\!\top \circ X^{\cup}) \\
= & \qquad \{ \;\; R \circ X \circ R^{\cup} \neq \bot\!\!\bot \;\Rightarrow\; X^{\cup} \circ R^{\cup} \circ R \circ X \neq \bot\!\!\bot, \text{ cone axiom} \;\; \} \\
 & \bigsqcup(X \;:\; X \in B \;:\; X \circ \top\!\!\top \circ X^{\cup}) \\
= & \qquad \{ \;\; \text{def "}\in\text{"} \;\; \} \\
 & \bigsqcup(X \;:\; X \in B \;:\; X) \\
= & \qquad \{ \;\; \text{extensionality} \;\; \} \\
 & B
\end{array}
$$

Let us consider now injectivity and surjectivity While in a purely functional setting, these notions are specialisations of the notion of functionality, in a relational setting, it is natural that they exist as general notions in their own right, injectivity being dual to functionality, and surjectivity being dual to totality. This duality may be seen directly in the summary of our six typing notions given below.

$$
\begin{array}{lcl}
R \in A \sim B & \mathrel{\widehat{=}} & A \circ R = R = R \circ B \\[1ex]
f \in A \leftarrow B & \mathrel{\widehat{=}} & f \in A \sim B \;\;\wedge\;\; A \sqsupseteq f \circ f^{\cup} \\[1ex]
f \text{ is } \textit{functional to } A & \mathrel{\widehat{=}} & A \circ f = f \;\;\wedge\;\; A \sqsupseteq f \circ f^{\cup} \\[1ex]
f \text{ is } \textit{injective on } B & \mathrel{\widehat{=}} & f \circ B = f \;\;\wedge\;\; B \sqsupseteq f^{\cup} \circ f \\[1ex]
R \text{ is } \textit{total on } B & \mathrel{\widehat{=}} & R \circ B = R \;\;\wedge\;\; R^{\cup} \circ R \sqsupseteq B \\[1ex]
R \text{ is } \textit{surjective to } A & \mathrel{\widehat{=}} & A \circ R = R \;\;\wedge\;\; R \circ R^{\cup} \sqsupseteq A
\end{array}
$$

We conclude with some useful properties of the "$\leftarrow$" operator:

$$
\begin{array}{llcl}
\text{(a)} & f \circ g \;\in\; A \leftarrow D & \Leftarrow & f \in A \leftarrow B \;\;\wedge\;\; g \in C \leftarrow D \;\;\wedge\;\; B \sqsupseteq C \\[1ex]
\text{(b)} & f \in A \leftarrow B & \Leftarrow & f \in C \leftarrow B \;\;\wedge\;\; C \subseteq A \\[1ex]
\text{(c)} & f \in A \leftarrow B & \Leftarrow & f \in A \leftarrow C \;\;\wedge\;\; C \lhd B \\[1ex]
\text{(d)} & A \circ f \;\in\; A \leftarrow B & \Leftarrow & f \in C \leftarrow B \;\;\wedge\;\; A \lhd C \\[1ex]
\text{(e)} & A \circ f \;\in\; A \leftarrow B & \Leftarrow & f \in C \leftarrow B \;\;\wedge\;\; A \sqsupseteq C \\[1ex]
\text{(f)} & f \circ B \;\in\; A \leftarrow B & \Leftarrow & f \in A \leftarrow C \;\;\wedge\;\; B \subseteq C
\end{array}
$$

Some comments: (a) also holds if arrows f and g are total, and if both are injective or surjective if $B = C$; (b) also holds if both arrows are injective or total; (c) also holds for surjective arrows, for total arrows if $C \mathrel{\lhd\!|} B$, and for injective arrows if $C \subseteq B$; (d) also holds for surjective arrows; (e) also holds for total arrows and for injective arrows; (f) also holds for total arrows, and for injective arrows if $C \subseteq B$.

Chapter 4

Defining new types

In calculational approaches to type theory (where calculational elegance rather than a type inference algorithm is the goal), it has become standard to adopt (variations of) the "Hagino" or, as we prefer to call it, *F–algebra* paradigm of type definition. Good explanations are given by Malcolm [2] and Fokkinga and Meijer [14]. Although we ourselves do not adopt directly the F–algebra paradigm, it provides a stepping stone to the "real–fixpoint" approach introduced later on in this chapter. We give now a brief review of the F–algebra approach: it is not intended as a tutorial; readers not familiar with this this paradigm are directed to [2] and [14].

As in most modern functional languages, types in the F–algebra style are built via constructor functions. For example, the type of cons–lists $A*$ over some base type A may be built from a null–ary function $nil \in A* \leftarrow \mathbb{1}$ and a binary function $cons \in A* \leftarrow A \times A*$. Implicit is that the type $A*$ so defined is the "least" type induced by the constructors: nothing is in $A*$ besides those elements generated by application of the constructor functions *nil* and *cons*.

In the F–algebra paradigm of type definition, we supply not the constructor functions themselves, but only their type, by means of the categorical notion of a "functor". The defined type itself, and constructors for the type, are then given by a so–called "initial algebra" over the functor, initiality capturing precisely the sense in which the defined type is the least induced by the constructors. Recognising and exploiting this initiality is a key step to concise and elegant calculation involving datatypes. Let us now quickly review the technical details.

Given an endofunctor F, an algebra over F, more commonly called an *F–algebra*, is an arrow $f : F.A \rightarrow A$, for some object A called the *carrier* of the algebra. F–algebras form objects in a category $\mathsf{F\text{–}alg}$, where a morphism between algebras $f : F.A \rightarrow A$ and $g : F.B \rightarrow B$, called an *F–homomorphism*, is given by an arrow $h : A \rightarrow B$ for which $h \circ f = g \circ F.h$.

An object A in a category is called *initial* if there is precisely one arrow from A to B for each object B. Let $in : F.\mu F \rightarrow \mu F$ be initial in the category $\mathsf{F\text{–}alg}$ (the use of "μ" will be explained shortly.) The F–homomorphisms which witness the initiality of the algebra "*in*" are denoted using a special bracketing notation $(\!|\text{–}|\!)$, and called *catamorphisms*. More precisely, for each F–algebra f, we denote the unique F–homomorphism from *in* to f by $(\!|f|\!)$. Catamorphisms are nothing more than generalisations to arbitrary datatypes of the familiar "reduce" operator (or "fold" as it is sometimes called) on cons–lists from functional programming.

A fixpoint of an endofunctor F is an object A for which $F.A$ is isomorphic to A. An initial F–algebra $in : F.\mu F \to \mu F$ is a fixpoint of functor F, in that $in \cong F.in$ in the category $\mathsf{F\text{–}alg}$. (Note the implicit type conversions in this theorem: F acts upon "in" as an arrow in the base category, yielding an arrow $F.in : F.(F.\mu F) \to F.\mu F$, then regarded again as an object (i.e. F–algebra) in category $\mathsf{F\text{–}alg}$.) We find that "in" itself forms one half of the isomorphism, while the catamorphism $(\!|F.in|\!)$, often denoted by "out", forms the other.

Since $\mathsf{F\text{–}alg}$ is built upon some base category, C say, (i.e. every arrow in $\mathsf{F\text{–}alg}$ is an arrow in C), an immediate corollary of the isomorphism $in \cong F.in$ in $\mathsf{F\text{–}alg}$ is an isomorphism $\mu F \cong F.\mu F$ in C. We call the carrier of "in", denoted by μF, the *least fixpoint type* induced by F, while "in" itself serves as a *constructor* for the type μF. (Conversely, the catamorphism $out = (\!|F.in|\!)$ serves as a *destructor* for μF.)

Fundamental to a calculation theory of types based on the F–algebra paradigm of type definition is the so–called *unique extension property* for catamorphisms, normally abbreviated by "UEP". It captures precisely that $(\!|f|\!)$ is the unique solution in X to the recursion equation $X \circ in \;=\; f \circ F.X$:

Definition 62: (UEP for functional catamorphisms)

$$X \circ in \;=\; f \circ F.X \quad \equiv \quad X = (\!|f|\!)$$

The UEP provides a basis for concise and elegant equational proofs involving catamorphisms, avoiding the ritual steps of "proof by induction". (Note above that f is required to be an F–algebra, i.e. an arrow $F.A \to A$ for some type A. An important advantage of the so–called "real fixpoint" approach to types presented later on is that no such type information is required in building catamorphisms.)

4.1 Relators

In the F–algebra paradigm, datatypes are defined by means of an endofunctor. In this section, we introduce the category "difun" of difunctional specs, and explore the notion of an endofunctor on this category.

Definition 63: (The category "difun")

objects:	partial equivalence relations
arrows:	typed total difunctionals $f \in A \leftarrow B$
composition:	the "$\circ$" operator on specs
identities:	pers are their own identity arrows

A category consists of two parts: objects and arrows. A structure preserving map between categories, a "functor", consists then of two functions (both denoted for convenience by the same name), one mapping objects to objects, and one mapping arrows to arrows. The categorical structure is preserved by a functor F in that an identity arrow id_A is mapped to the identity $id_{F.A}$, an arrow $f : A \to B$ is mapped to $F.f : F.A \to F.B$, and a composite arrow $f \circ g$ is mapped to $F.f \circ F.g$.

Since id_A in **difun** is just the per A itself, preservation of identity objects, $F.id_A = id_{F.A}$, holds automatically for any function F which maps pers to pers. Moreover, since objects and arrows are both special kinds of specs in **difun**, a functor F : **difun** $\to$ **difun** may be viewed just as a single function F from specs to specs, preserving three notions: objects, arrows, and composition. We require then that:

$$
\begin{array}{ll}
(1) & per.(F.A) \;\Leftarrow\; per.A \\
(2) & F.f \in F.A \leftarrow F.B \;\Leftarrow\; f \in A \leftarrow B \\
(3) & F.(f \circ g) \;=\; F.f \circ F.g
\end{array}
$$

The category **difun** has extra structure beyond that required for being a category. In particular, it has a partial–order on homsets (it is order–enriched) corresponding to inclusion of relations, and an operator "$\cup$" which flips arrows around (it is in fact self–dual, since $difun.f \;\equiv\; difun.f\cup$.) It is natural that this extra structure should be preserved. This leads us to introduce the notion of a *relator*: an endofunctor on **difun** which, in addition to its functorial properties, is required to preserve the poset and reverse structure of relations.

Since the notions "*per*" and "$\leftarrow$" are not primitive, it seems likely that we can find simpler properties which guarantee the first two properties required of F above. In fact, the proofs below show respectively that the two extra conditions required for a functor F being a relator are in themselves sufficient to entail both properties:

Proof

$$
\begin{array}{ll}
 & per.(F.A) \\
\equiv & \qquad \{\ \text{def per}\ \} \\
 & F.A \;=\; F.A \circ (F.A)\cup \\
\equiv & \qquad \{\ \text{assume: } F.(R\cup) = (F.R)\cup\ \} \\
 & F.A \;=\; F.A \circ F.(A\cup) \\
\equiv & \qquad \{\ \text{distribution}\ \} \\
 & F.A \;=\; F.(A \circ A\cup) \\
\Leftarrow & \qquad \{\ \text{Leibniz}\ \} \\
 & A \;=\; A \circ A\cup \\
\equiv & \qquad \{\ \text{def per}\ \} \\
 & per.A
\end{array}
$$

Proof

$$
\begin{array}{ll}
 & F.f \in F.A \leftarrow F.B \\
\equiv & \qquad \{\ \text{def} \leftarrow\ \} \\
 & F.A \circ F.f \;=\; F.f \;=\; F.f \circ F.B \;\wedge \\
 & \qquad F.A \sqsupseteq F.f \circ (F.f)\cup \;\wedge\; (F.f)\cup \circ F.f \sqsupseteq F.B
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \quad \{ \text{ assume: } F.(R\cup) = (F.R)\cup \text{, distribution } \} \\
& F.(A \circ f) \;=\; F.f \;=\; F.(f \circ B) \;\;\wedge \\
& \quad F.A \;\sqsupseteq\; F.(f \circ f\cup) \;\;\wedge\;\; F.(f\cup \circ f) \;\sqsupseteq\; F.B \\
\Leftarrow & \quad \{ \text{ assume: F.R} \sqsupseteq \text{F.S} \Leftarrow \text{R} \sqsupseteq \text{S } \} \\
& A \circ f = f = f \circ B \;\;\wedge\;\; A \;\sqsupseteq\; f \circ f\cup \;\;\wedge\;\; f\cup \circ f \;\sqsupseteq\; B \\
\equiv & \quad \{ \text{ def} \leftarrow \ \} \\
& f \;\in\; A \leftarrow B
\end{array}
$$

We have shown that for a function F that preserves the "$\sqsupseteq$" and "$\cup$" operators on specs, all that is required extra to ensure that F be functorial is that it preserve the "$\circ$" operator. We now make concrete our notion of "relator":

Definition 64: A *relator* is a function F from specs to specs for which

(1) $F.R \sqsupseteq F.S \;\Leftarrow\; R \sqsupseteq S$
(2) $F.(R\cup) = (F.R)\cup$
(3) $F.(R \circ S) \;=\; F.R \circ F.S$

We always use capital letters $F, G, H, \ldots$ to denote relators. We often apply the commutativity law $F.(R\cup) = (F.R)\cup$ silently in calculation, writing simply $F.R\cup$ without parenthesis. By definition, every relator is a functor. Every functor is not however a relator. An example is given by $F.X = \top\!\!\top \circ (X \circ X\cup \circ X \;\sqcap\; \neg X) \circ \top\!\!\top$, mapping difunctionals to $\perp\!\!\perp$ and all other specs to $\top\!\!\top$. While F is indeed functorial, it is not monotonic, e.g. $F.\top\!\!\top = \perp\!\!\perp$ and $F.\neg I = \top\!\!\top$, and is hence is not a relator. We shall see however that the important class of "polynomial functors" (roughly speaking, those built from the disjoint sum and cartesian product functors) do indeed satisfy the extra requirements for being a relator.

The three properties above define a "unary" relator. The notion can be generalised in the obvious way to "n–ary" relators. With a view to defining binary relators corresponding to cartesian product and disjoint sum, we give below conditions for a binary function "$\otimes$" on specs being a binary relator:

(1) $R\cup \otimes S\cup \;=\; (R \otimes S)\cup$
(2) $R \otimes S \sqsupseteq V \otimes W \quad\Leftarrow\quad R \sqsupseteq V \wedge S \sqsupseteq W$
(3) $(R \circ S) \otimes (V \circ W) \;=\; R \otimes V \circ S \otimes W$

(*Aside*: There are six primitive operators defined on specs, only three of which relators are required to preserve. Why not preserve the "$\sqcup$", "$\sqcap$", and "$\neg$" operators also ? Two observations: preservation of "$\sqcup$" (or "$\sqcap$") subsumes the monotonicity requirement of relators; preservation of "$\neg$" follows from that of "$\sqcup$" and "$\sqcap$".

We note first that preserving finite (and hence arbitrary) lubs and glbs is out of the question. In particular, preserving the empty set of specs under "$\sqcup$" and "$\sqcap$" requires that $F.\perp\!\!\perp = \perp\!\!\perp$ and $F.\top\!\!\top = \top\!\!\top$; either of these identifications would prevent us defining relators of the form $F.X = A$ (for A a per), so–called "constant relators". (These are important because they form a basis for building polynomial relators.) What however about just preserving binary lubs and glbs ? Some relators in fact do preserve these (e.g. constant relators and the identity relator), but in general again, polynomials relators do not. *End of Aside*)

Polynomial relators

In the previous section we reviewed the definition of types in terms of fixpoints of functors, the so–called "F–algebra" paradigm. In the next section we introduce the "real–fixpoint" paradigm, where types are defined as fixpoints of relators. We introduce now the *polynomial relators*, a class of relators in terms of which we can define "sum of product" types as found in most functional languages. We present only the definitions and the main results; for proofs, further explanation, and an account of the special "map relators", the reader is referred to [3].

Definition 65: Each of the following are *polynomial relators*:

$$
\begin{array}{lll}
(1) & \mathcal{I}.X & \widehat{=} \;\; X \\
(2) & \hat{A}.X & \widehat{=} \;\; A \\
\\
(3) & X \times Y & \widehat{=} \;\; (X \circ \ll) \vartriangle (Y \circ \gg) \\
(4) & X + Y & \widehat{=} \;\; (\hookrightarrow \circ X) \triangledown (\hookleftarrow \circ Y) \\
\\
(5) & (F \,\hat{\otimes}\, G).X & \widehat{=} \;\; F.X \otimes G.X
\end{array}
$$

(The reader may like to be reminded that, while functors must be defined separately on objects and arrows, a relator has only one part, the definition on specs.) Some comments: $\mathcal{I}$ is the identity relator; $\hat{A}$ is the constant relator which always returns per A; "$\times$" is a binary relator corresponding to cartesian product; "$+$" corresponds to disjoint sum. Finally, "$\hat{\otimes}$" is a "lifted" version of a binary relator "$\otimes$" which acts not on specs, but on relators, i.e. for unary relators F and G, $F \hat{\otimes} G$ is itself a relator. Clearly in (5) we require that all relators involved be polynomial.

Simple examples of polynomial relators are $\hat{\top\!\top} \mathbin{\hat{+}} \mathcal{I}$ and $\hat{A} \mathbin{\hat{+}} (\mathcal{I} \mathbin{\hat{\times}} \mathcal{I})$. (The first gives rise to a type of natural numbers, the second to non–empty binary trees.) Eliminating the lifting, these may be written simply as $F.X = \top\!\top + X$ and $F.X = A + (X \times X)$. Important note: in our per–based theory, "$\top\!\top$" is a type with precisely one element, a so–called *unit type*. In the monotypes world, no such convenient spec exists already; we are required to postulate the existence of a unit spec by means of an extra axiom. (See [3] for the details.)

The definitions given above for "$\times$" and "$+$" are familiar from category theory, except for our choice of symbols: "$\ll$" and "$\gg$" are specs pronounced respectively as "project left" and "project right"; specs "$\hookrightarrow$" and "$\hookleftarrow$" are pronounced "inject left" and "inject right". It is important to note that we have only two projection specs and two injection specs: our theory is in a sense *truly* polymorphic; we don't have separate projection and injection specs for each type.

The following diagrams are useful in understanding our choice of notation:

$$X \;\ll\; X \times Y \;\gg\; Y$$

$$X \;\hookrightarrow\; X + Y \;\hookleftarrow\; Y$$

As is now standard in calculational approaches to type theory, we adopt the infix symbol "$\vartriangle$" (pronounced "split") in place of the rather verbose $\langle -,-\rangle$ notation from category theory; dually, we use "$\triangledown$" (pronounced "junc", abbreviating junction) in place of $[-,-]$. Split and junc have the following pointwise interpretation:

$$
\begin{aligned}
(x,y)\ (R \vartriangle S)\ z \quad &\equiv \quad x\ R\ z \ \wedge\ y\ S\ z \\
x\ (R \triangledown S)\ (\hookrightarrow.y) \quad &\equiv \quad x\ R\ y \\
x\ (R \triangledown S)\ (\hookleftarrow.y) \quad &\equiv \quad x\ S\ y
\end{aligned}
$$

We find however that, working in a relational framework, "$\vartriangle$" and "$\triangledown$" can be defined in terms of the projection and injection specs respectively:

Definition 66: $R \vartriangle S \quad \widehat{=} \quad (\ll^{\cup} \circ R)\ \sqcap\ (\gg^{\cup} \circ S)$

Definition 67: $R \triangledown S \quad \widehat{=} \quad (R \circ \hookrightarrow^{\cup})\ \sqcup\ (S \circ \hookleftarrow^{\cup})$

The projection and injection specs themselves are not defined explicitly, but are required to satisfy the three axioms given below, from which all properties we expect of these specs and those defined in terms thereof can be verified.

Axiom 68: $(R \circ \ll \sqcap S \circ \gg)\ \circ\ (\ll^{\cup} \circ T\ \sqcap\ \gg^{\cup} \circ U) \quad = \quad R \circ T \sqcap S \circ U$

Axiom 69: $\top\!\!\top \circ \ll\ =\ \top\!\!\top \circ \gg$

Axiom 70: $(R \circ \hookrightarrow^{\cup} \sqcup S \circ \hookleftarrow^{\cup})\ \circ\ (\hookrightarrow \circ T \sqcup \hookleftarrow \circ U) \quad = \quad R \circ T \sqcup S \circ U$

From the above definitions and axioms, we can show that $\mathcal{I}$, $\hat{A}$, $F\hat{\otimes}G$, "$\times$", and "$+$" indeed satisfy the three properties required of a relator. Moreover, we have

Computation rules

$$
\begin{aligned}
\ll \circ (R \vartriangle S) \ &=\ R \circ S^{>} \\
\gg \circ (R \vartriangle S) \ &=\ S \circ R^{>}
\end{aligned}
$$

$$
\begin{aligned}
(R \triangledown S) \circ \hookrightarrow \ &=\ R \\
(R \triangledown S) \circ \hookleftarrow \ &=\ S
\end{aligned}
$$

Fusion rules

$$
\begin{aligned}
R \times S \circ T \vartriangle U \ &=\ (R \circ T) \vartriangle (S \circ U) \\
R \times S \circ T \times U \ &=\ (R \circ T) \times (S \circ U)
\end{aligned}
$$

$$
\begin{aligned}
R \triangledown S \circ T + U \ &=\ (R \circ T) \triangledown (S \circ U) \\
R + S \circ T + U \ &=\ (R \circ T) + (S \circ U)
\end{aligned}
$$

Other properties

$$
\begin{array}{lll}
\hookrightarrow^{\cup} \circ \hookrightarrow \ = \ I & \qquad & \ll \circ \ll^{\cup} \ = \ I \\
\hookleftarrow^{\cup} \circ \hookleftarrow \ = \ I & & \gg \circ \gg^{\cup} \ = \ I \\
\hookrightarrow^{\cup} \circ \hookleftarrow \ = \ \perp\!\!\perp & & \ll \circ \gg^{\cup} \ = \ \top\!\!\top \\
\hookleftarrow^{\cup} \circ \hookrightarrow \ = \ \perp\!\!\perp & & \gg \circ \ll^{\cup} \ = \ \top\!\!\top \\
\hookrightarrow \circ \hookrightarrow^{\cup} \ = \ I + \perp\!\!\perp & & \ll^{\cup} \circ \ll \ = \ I \times \top\!\!\top
\end{array}
$$

4.2 Types as real–fixpoints

In the F–algebra paradigm of type definition, the so–called "least fixpoint type" μF satisfies an isomorphism $\mu F \cong F.\mu F$, as witnessed by the constructor function $in : F.\mu F \rightarrow \mu F$, and destructor function $out : \mu F \rightarrow F.\mu F$. If however we work in a relational setting, with F as a relator rather than just a functor, we can take the "real" least fixpoint of F, gaining a true equality $\mu F = F.\mu F$, rather than just an isomorphism. This deviation from the standard F–algebra paradigm is due to Backhouse [3], where it is used in extending the (functional) notion of catamorphism to relations. (We summarise the construction of relational catamorphisms at the end of this section. Mention should also be made of Fokkinga and Meijer [14], whome adopt a real fixpoint approach to types in a cpo–based functional world.)

Since a relator F is by definition monotonic, the existence of a lattice of its fixpoints is predicted by the Knaster–Tarski theorem. We denote the least and greatest fixpoints of a relator F respectively by μF and νF. Our first result is that both μF and νF are pers, allowing us to refer to them as fixpoint "types".

Lemma 71: $per.\mu F \;\wedge\; per.\nu F$

(Recall for the proofs below the induction laws for least and greatest fixpoints: $X \sqsupseteq \mu F \;\Leftarrow\; X \sqsupseteq F.X$ and $\nu F \sqsupseteq X \;\Leftarrow\; F.X \sqsupseteq X$.)

Proof (μF is symmetric; similarly for νF)

$$
\begin{array}{ll}
 & (\mu F)^{\cup} = \mu F \\
\equiv & \qquad \{ \text{ reverse } \} \\
 & (\mu F)^{\cup} \sqsupseteq \mu F \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & (\mu F)^{\cup} \sqsupseteq F.((\mu F)^{\cup}) \\
\equiv & \qquad \{ \text{ relators } \} \\
 & (\mu F)^{\cup} \sqsupseteq (F.\mu F)^{\cup} \\
\equiv & \qquad \{ \ \mu F = F.\mu F \ \} \\
 & true
\end{array}
$$

Proof (μF is transitive)

$$
\begin{array}{ll}
 & \mu F \;\sqsupseteq\; \mu F \circ \mu F \\
\equiv & \qquad \{ \text{ factors } \} \\
 & \mu F \;/\; \mu F \;\sqsupseteq\; \mu F \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & \mu F \;/\; \mu F \;\sqsupseteq\; F.(\mu F \;/\; \mu F) \\
\Leftarrow & \qquad \{ \text{ lemma below } \} \\
 & \mu F \;/\; \mu F \;\sqsupseteq\; F.\mu F \;/\; F.\mu F \\
\equiv & \qquad \{ \ \mu F = F.\mu F \ \} \\
 & true
\end{array}
$$

Proof (νF is transitive)

$$
\begin{array}{ll}
 & \nu F \sqsupseteq \nu F \circ \nu F \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & F.(\nu F \circ \nu F) \sqsupseteq \nu F \circ \nu F \\
\equiv & \qquad \{ \text{ relators } \} \\
 & F.\nu F \circ F.\nu F \sqsupseteq \nu F \circ \nu F \\
\equiv & \qquad \{ \nu F = F.\nu F \} \\
 & true
\end{array}
$$

Used in the second proof above is a pseudo–distribution property of relators over the left factoring operator "/"; a similar result holds for "\":

Lemma 72:

$$F.R \,/\, F.S \sqsupseteq F.(R \,/\, S)$$

$$F.R \setminus F.S \sqsupseteq F.(R \setminus S)$$

We prove only the first result; the second proceeds similarly.

$$
\begin{array}{ll}
 & F.R \,/\, F.S \sqsupseteq F.(R \,/\, S) \\
\equiv & \qquad \{ \text{ factors } \} \\
 & F.R \sqsupseteq F.(R \,/\, S) \circ F.S \\
\equiv & \qquad \{ \text{ relators } \} \\
 & F.R \sqsupseteq F.(R \,/\, S \circ S) \\
\Leftarrow & \qquad \{ \text{ relators } \} \\
 & R \sqsupseteq R \,/\, S \circ S \\
\equiv & \qquad \{ \text{ cancellation } \} \\
 & true
\end{array}
$$

In addition to μF being the least fixpoint of F under the "$\sqsubseteq$" ordering on specs, we find that it is in fact also the least fixpoint of F under the "$\subseteq$" ordering on pers:

Lemma 73: $\mu F \subseteq A \;\Leftarrow\; A = F.A \,\wedge\, per.A$

(Note: the assumption $per.A$ is not used within the proof below; it is required only to allow A to be used as an argument to the subtype ordering "$\subseteq$".)

Proof

$$
\begin{array}{ll}
 & \mu F \subseteq A \\
\equiv & \qquad \{ \text{ def} \subseteq \} \\
 & \mu F \lhd A \,\wedge\, A \sqsupseteq \mu F \\
\equiv & \qquad \{ \text{ def} \lhd \} \\
 & \mu F \circ A = \mu F \;\wedge\; A \sqsupseteq \mu F \\
\equiv & \qquad \{ \text{ calculus } \} \\
 & \mu F \circ A \sqsupseteq \mu F \;\wedge\; \mu F \sqsupseteq \mu F \circ A \;\wedge\; A \sqsupseteq \mu F
\end{array}
$$

We proceed now to demonstrate each of the final conjuncts in turn:

Proof

$$
\begin{array}{ll}
 & \mu F \circ A \sqsupseteq \mu F \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & \mu F \circ A \sqsupseteq F.(\mu F \circ A) \\
\equiv & \qquad \{ \text{ relators } \} \\
 & \mu F \circ A \sqsupseteq F.\mu F \circ F.A \\
\equiv & \qquad \{ \ \mu F = F.\mu F, \ A = F.A \ \} \\
 & \mathit{true}
\end{array}
$$

Proof

$$
\begin{array}{ll}
 & \mu F \sqsupseteq \mu F \circ A \\
\equiv & \qquad \{ \text{ factors } \} \\
 & \mu F \,/\, A \sqsupseteq \mu F \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & \mu F \,/\, A \sqsupseteq F.(\mu F \,/\, A) \\
\Leftarrow & \qquad \{ \text{ lemma: } F.R \,/\, F.S \sqsupseteq F.(R \,/\, S) \ \} \\
 & \mu F \,/\, A \sqsupseteq F.\mu F \,/\, F.A \\
\equiv & \qquad \{ \ \mu F = F.\mu F, \ A = F.A \ \} \\
 & \mathit{true}
\end{array}
$$

Proof

$$
\begin{array}{ll}
 & A \sqsupseteq \mu F \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & A \sqsupseteq F.A \\
\equiv & \qquad \{ \ A = F.A \ \} \\
 & \mathit{true}
\end{array}
$$

In the F–algebra approach, the catamorphism $(\![f]\!)$ is given by the unique solution in X to the equation $X \circ in = f \circ F.X$, where "in" is an initial algebra over functor F. In the real fixpoint approach to types, Backhouse [3] makes the following definition: $(\![R]\!)$ is the least solution in X to the equation $X = R \circ F.X$. (No analogue to the "in" part in the F–algebra equation for catamorphisms is needed here, due to our having a true equality rather than just an isomorphism.)

As mentioned earlier, the so–called "unique extension property" (UEP) is fundamental to concise and elegant equational proofs involving catamorphisms. This property requires that $(\![R]\!)$ be expressible as the unique solution to some equation. In the relational approach, the defining equation for catamorphisms, $X = R \circ F.X$, has many solutions, of which $(\![R]\!)$ is the least. Backhouse gains a UEP for relational catamorphisms by showing that the stronger equation $X = R \circ F.X \circ \mu F$ has $(\![R]\!)$ as not just its least solution, but in fact its unique solution.

Definition 74: (UEP for relational catamorphisms)

$$X = (\![R]\!) \;\equiv\; X = R \circ F.X \circ \mu F$$

There are a number of points worth noting about the real–fixpoint paradigm of type definition. First of all, calculation with a true equality is more convenient than with an isomorphism. Secondly, as we shall see later, it is easier to generalise to patterns of recursion other than that exhibited by catamorphisms. Finally, whereas in the F–algebra approach we required in building a catamorphism $(\![f]\!)$ that f be an arrow $F.A \rightarrow A$ for some type A, in the relational approach, we require no such type information, our UEP being defined for arbitrary spec R. (Such elimination of type information is in fact one of the major goals of [3].)

4.3 Constructors and inductive types

We call a spec "*in*" that is difunctional and surjective to a per A a *constructor* for A. Difunctionality encodes the bijectivity property we expect of constructors; surjectivity ensures that every element in A can be made using in. Expanding the two requirements on in gives two equations: $A \circ in = in$ and $A = in \circ in^{\cup}$. Choosing A itself for in satisfies both equations: pers can serve as their own constructors.

Most interesting types have more than one constructor. With the type of non–empty lists for example, we have $[-]$ (making singleton lists) and "$+\!\!+$" (joining two lists together). We call a type with more than one constructor a *sum–type*. This notion has a simple axiomatisation; we give the definition for the binary case:

Definition 75: per A is a *sum–type* $\;\widehat{=}\;$ $A \circ I + I = A$

(With A being a per and "+" a relator, we see that $I + I \circ A = A$ is an equivalent definition.) Under the assumption that A is a sum–type, we seek now to break the constructor in (i.e. A itself) into two simpler constructors, g and h say, with $g \triangledown h = in$. Naturally, we require also that g and h both be difunctional to A.

Lemma 76: The assignments $g := A \circ \hookrightarrow$ and $h := A \circ \hookleftarrow$ satisfy our requirements, where "$\hookrightarrow$" and "$\hookleftarrow$" are the polymorphic injections in terms of which the sum–relator "+" is defined.

Let us show first of all that $g \triangledown h = A$:

$$
\begin{array}{ll}
 & g \triangledown h \\
= & \quad \{ \text{ def } g,\ h \ \} \\
 & (A \circ \hookrightarrow) \;\triangledown\; (A \circ \hookleftarrow) \\
= & \quad \{ \text{ def } \triangledown \ \} \\
 & A \circ \hookrightarrow \circ \hookrightarrow^{\cup} \sqcup A \circ \hookleftarrow \circ \hookleftarrow^{\cup} \\
= & \quad \{ \text{ distribution } \} \\
 & A \circ (\hookrightarrow \circ \hookrightarrow^{\cup} \sqcup \hookleftarrow \circ \hookleftarrow^{\cup})
\end{array}
$$

$$
\begin{array}{ll}
= & \quad \{ \text{ def } \triangledown \ \} \\
& A \circ (\hookrightarrow \triangledown \hookleftarrow) \\
= & \quad \{ \text{ def } + \ \} \\
& A \circ I{+}I \\
= & \quad \{ \ \mathit{sum\text{-}type}.A \ \} \\
& A
\end{array}
$$

It remains to show that g and h are difunctional to A. We prove the g result by way of example. It breaks into two parts: $A \circ g = g$ and $g \circ g^{\cup} \sqsubseteq A$.

Proof

$$
\begin{array}{ll}
& A \circ g \\
= & \quad \{ \text{ def } g \ \} \\
& A \circ A \circ \hookrightarrow \\
= & \quad \{ \ \mathit{per}.A \ \Rightarrow \ A \circ A = A \ \} \\
& A \circ \hookrightarrow \\
= & \quad \{ \text{ def } g \ \} \\
& g
\end{array}
$$

Proof

$$
\begin{array}{ll}
& g \circ g^{\cup} \\
\sqsubseteq & \quad \{ \text{ calculus } \} \\
& g \circ g^{\cup} \ \sqcup \ h \circ h^{\cup} \\
= & \quad \{ \text{ injections axiom } \} \\
& (g \circ \hookrightarrow^{\cup} \ \sqcup \ h \circ \hookleftarrow^{\cup}) \circ (\hookrightarrow \circ g^{\cup} \ \sqcup \ \hookleftarrow \circ h^{\cup}) \\
= & \quad \{ \text{ def } \triangledown \ \} \\
& (g \triangledown h) \circ (g \triangledown h)^{\cup} \\
= & \quad \{ \text{ shown above } \} \\
& A \circ A^{\cup} \\
= & \quad \{ \ \mathit{per}.A \ \} \\
& A
\end{array}
$$

Some simple types can be defined directly. For example, $\top\!\!\top + \top\!\!\top$ may be viewed as a type of booleans. Most of our types however are defined in terms of a relator. The basic method is to take a fixpoint of a relator F, yielding a per A satisfying $F.A = A$. For such *fixpoint types* we find that $\mathit{in} \in A \leftarrow F.A$. All our types however are not fixpoint types. Given a type A we can make a new type $B \lhd A$ by imposing laws and restrictions on the constructors of A. In general, A being a fixpoint type under F does not ensure that such a B is also. For the typing $\mathit{in} \in A \leftarrow F.A$ to be valid for an arbitrary per A we find that, $A \circ F.A = A$ must hold true. Using the "$\lhd$" ordering on pers, this equation may be expressed as $A \lhd F.A$.

Definition 77: per A is $F\text{-}\mathit{inductive} \ \ \hat{=} \ \ A \lhd F.A$

As its name suggests, an inductive type is one whose subcomponents of every element are themselves elements of the type. The inductivity property is more general than the fixpoint property: $A = F.A \;\Rightarrow\; A \lhd F.A$, but the reverse does not in general hold. Provided then that $A \lhd F.A$ holds, A may serve as its own constructor.

In our being able to handle types with laws and restrictions, we do not expect in general that the constructor "*in*" be either injective or total on $F.A$: with laws it is possible that an element can be made in more than one way; with restrictions it is possible that some elements cannot be made at all. In this context, two strengthenings of the notion of F–inductivity have special meaning: $A \subseteq F.A$ expresses that the constructor for A is *injective*; $A \lhd\!\mid F.A$ expresses that constructor for type A is *total*. The first tells us that no laws have been applied in building A, the second that no restrictions have been applied.

Lemma 78:

$$\text{"}\mathit{in}\text{" is injective on } F.A \;\;\equiv\;\; A \subseteq F.A$$

$$\text{"}\mathit{in}\text{" is total on } F.A \;\;\equiv\;\; A \lhd\!\mid F.A$$

We only prove the first of these results; the second proceeds similarly.

$$\begin{array}{ll}
 & \text{"}\mathit{in}\text{" injective on } F.A \\
\equiv & \qquad \{\ \mathit{in} = A,\ \text{def injective}\ \} \\
 & A \circ F.A \;=\; A \;\;\wedge\;\; F.A \sqsupseteq A^{\cup} \circ A \\
\equiv & \qquad \{\ \text{def} \lhd,\;\; \mathit{per}.A \;\equiv\; A^{\cup} \circ A \;=\; A\ \} \\
 & A \lhd F.A \;\;\wedge\;\; F.A \sqsupseteq A \\
\equiv & \qquad \{\ \text{def} \subseteq\ \} \\
 & A \subseteq F.A
\end{array}$$

In practice, the relator F will always be a *sum–relator*. Let us consider, for example, the case where F is expressible as a sum $F = G \hat{+} H$ of relators G and H, i.e. $\forall\,(X :: F.X = G.X + H.X)$. Naturally, we find that the notion of a per being inductive with respect to a sum–relator is stronger than it being a sum–type:

Lemma 79: per A is $G \hat{+} H$ inductive $\;\Rightarrow\;$ A is a sum–type

Proof

$$\begin{array}{ll}
 & A \;\circ\; I + I \\
= & \qquad \{\ \text{inductivity}\ \} \\
 & A \;\circ\; G.A + H.A \;\circ\; I + I \\
= & \qquad \{\ \text{def} +,\ \text{fusion}\ \} \\
 & A \;\circ\; G.A + H.A \\
= & \qquad \{\ \text{inductivity}\ \} \\
 & A
\end{array}$$

We found earlier that for A a sum–type, $g = A \circ \hookrightarrow$ and $h = A \circ \hookleftarrow$ serve as constructors for A. The lemma above tells us that these also work as constructors for a type which is inductive with respect to a sum–relator. We must check however that in this use, g works upon elements of $G.A$, and h upon elements of $H.A$, allowing us to write $g \in A \leftarrow G.A$ and $h \in A \leftarrow H.A$. Let us show the first of these:

$$
\begin{array}{cl}
 & g \circ G.A \\
= & \quad \{ \ \text{def } g \ \} \\
 & A \circ \hookrightarrow \circ G.A \\
= & \quad \{ \ \text{injections} \ \} \\
 & A \circ (G.A + H.A) \circ \hookrightarrow \\
= & \quad \{ \ F = G \hat{+} H \ \} \\
 & A \circ F.A \circ \hookrightarrow \\
= & \quad \{ \ \mathit{inductive}.A \ \} \\
 & A \circ \hookrightarrow \\
= & \quad \{ \ \text{def } g \ \} \\
 & g
\end{array}
$$

Earlier we found that replacing "$\lhd$" in the definition of inductivity with the stronger notions of "$\subseteq$" and "$\lhd|$" were equivalent respectively to the constructor $in \in A \leftarrow F.A$ being injective and total. Under the assumption $F = G \hat{+} H$, we find that similar results hold for the constructors $g \in A \leftarrow G.A$ and $h \in A \leftarrow H.A$.

> **Lemma 80:** $A \lhd| F.A$ implies that g and h are total; $A \subseteq F.A$ implies that g and h are injective, and construct disjoint sets of elements in A.

On the totality and injectivity side, let us prove the injectivity result for g. Since $g \circ G.A = g$ is shown above, verifying $g^{\cup} \circ g \sqsubseteq G.A$ is all that is required:

$$
\begin{array}{cl}
 & g^{\cup} \circ g \\
= & \quad \{ \ \text{def } g \ \} \\
 & \hookrightarrow^{\cup} \circ A^{\cup} \circ A \circ \hookrightarrow \\
= & \quad \{ \ \mathit{per}.A \ \} \\
 & \hookrightarrow^{\cup} \circ A \circ \hookrightarrow \\
\sqsubseteq & \quad \{ \ A \subseteq F.A \ \Rightarrow \ A \sqsubseteq F.A \ \} \\
 & \hookrightarrow^{\cup} \circ F.A \circ \hookrightarrow \\
= & \quad \{ \ F = G \hat{+} H \ \} \\
 & \hookrightarrow^{\cup} \circ (G.A + H.A) \circ \hookrightarrow \\
= & \quad \{ \ \text{injections} \ \} \\
 & \hookrightarrow^{\cup} \circ \hookrightarrow \circ G.A \\
= & \quad \{ \ \text{injections} \ \} \\
 & G.A
\end{array}
$$

The disjointness result is proved as follows:

$$
\begin{array}{ll}
 & g{<} \;\sqcap\; h{<} \;=\; \perp\!\!\perp \\
\equiv & \quad \{ \text{ lemma 16 } \} \\
 & g{\cup} \circ h \;=\; \perp\!\!\perp \\
\equiv & \quad \{ \text{ def } g,h \ \} \\
 & \hookrightarrow{\cup} \circ A{\cup} \circ A \circ \hookleftarrow \;=\; \perp\!\!\perp \\
\equiv & \quad \{ \ per.A \ \} \\
 & \hookrightarrow{\cup} \circ A \circ \hookleftarrow \;=\; \perp\!\!\perp \\
\Leftarrow & \quad \{ \ A \subseteq F.A \Rightarrow A \sqsupseteq F.A \ \} \\
 & \hookrightarrow{\cup} \circ F.A \circ \leftarrow \;=\; \perp\!\!\perp \\
\equiv & \quad \{ \ F = G \hat{+} H \ \} \\
 & \hookrightarrow{\cup} \circ (G.A + H.A) \circ \hookleftarrow \;=\; \perp\!\!\perp \\
\equiv & \quad \{ \text{ injections } \} \\
 & \hookrightarrow{\cup} \circ \hookleftarrow \circ H.A \;=\; \perp\!\!\perp \\
\equiv & \quad \{ \ \hookrightarrow{\cup} \circ \hookleftarrow = \perp\!\!\perp \ \} \\
 & true
\end{array}
$$

We make, without proof, one final observation:

> **Lemma 81:** $A \lhd \mu F$ expresses that the type A is *finitely generated*, in that all elements of A can be built via a finite number of applications of its constructors. (Finite generation is important because it ensures that recursive operations over the type terminate.)

Let us summarise the notions introduced above:

$A \lhd F.A$	—	F–inductivity
$A \subseteq F.A$	—	injective constructors: no laws
$A \lhd\!\mid F.A$	—	total constructors: no restrictions
$A \lhd \mu F$	—	finite generation

In the F–algebra paradigm of type definition, the catamorphism $(\![f]\!)$ is defined as the unique solution in X to the recursion equation $X \circ in = f \circ F.X$, where "in" is an initial algebra over functor F. Substituting $X := (\![f]\!)$ in this equation gives the *computation rule* for catamorphisms, so–called because it provides a recursive scheme for computing $(\![f]\!)$. In our context we can be more general: rather than presenting a computation rule specifically for catamorphisms, we give sufficient conditions for the existence of such a rule for an arbitrary spec X.

> **Lemma 82:** (Computation Rule)
>
> $$X \circ A = X \quad\wedge\quad X = (R \triangledown S) \circ F.X$$
>
> $\Rightarrow$
>
> $$X \circ g = R \circ G.X \quad\wedge\quad X \circ h = S \circ H.X$$

It is easy to verify that, for arbitrary specs R and S, catamorphisms of the form $(\![R \triangledown S]\!)$ are examples of such a spec X. We prove only the first conjunct of the result of lemma 82; the second proceeds similarly.

$$
\begin{array}{ll}
 & X \circ g \\
= & \quad \{ \ \text{def}\ g \ \} \\
 & X \circ A \circ \hookrightarrow \\
= & \quad \{ \ X \circ A = X \ \} \\
 & X \circ \hookrightarrow \\
= & \quad \{ \ X = (R \triangledown S) \circ F.X, \ \ F = G \hat{+} H \ \} \\
 & (R \triangledown S) \circ (G.X + H.X) \circ \hookrightarrow \\
= & \quad \{ \ \text{fusion} \ \} \\
 & (R \circ G.X) \triangledown (S \circ H.X) \ \circ \hookrightarrow \\
= & \quad \{ \ \text{injections} \ \} \\
 & R \circ G.X
\end{array}
$$

4.4 Recursive specs

Recall that in the "real fixpoint" approach to datatypes, the catamorphism $(\![R]\!)$ is given by the least solution in X to the equation $X = R \circ F.X$, where R is a spec and F is a relator. We can generalise as follows: a spec is *right F–recursive* if a solution to the equation $X = R.X \circ F.X$, where $R.X$ is some spec–calculi expression, possibly involving X. We consider not just the least solutions to such equations, but arbitrary solutions. In particular, we are also interested in greatest solutions. Our motivation for considering F–recursive specs is that, in many situations, the assumption that a spec be F–catamorphic would be needlessly strong.

To show that a spec is right F–recursive, we are required to exhibit such an expression R. Calculation can easily become messy when existential arguments are involved. We find however that we can eliminate the "$\exists$", and indeed R itself:

Lemma 83: $\exists (R :: X = R.X \circ F.X) \ \equiv \ X = X/F.X \circ F.X$

The "$\Leftarrow$" proof is trivial: define $R.X$ by $X/F.X$. In the "$\Rightarrow$" direction, assuming $X = R.X \circ F.X$ for some R, we proceed by mutual containment:

Proof "$\sqsupseteq$"

$$
\begin{array}{ll}
 & X \ \sqsupseteq \ X/F.X \circ F.X \\
\equiv & \quad \{ \ \text{cancellation} \ \} \\
 & X \sqsupseteq X \\
\equiv & \quad \{ \ \text{assumption} \ \} \\
 & X \ \sqsupseteq \ R.X \ \circ \ F.X
\end{array}
$$

Proof "$\sqsubseteq$"

$$
\begin{array}{ll}
 & X/F.X \circ F.X \ \sqsupseteq \ X \\
\equiv & \quad \{ \ \text{assumption} \ \} \\
 & X/F.X \circ F.X \ \sqsupseteq \ R.X \ \circ \ F.X
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \quad \{ \text{ monotonicity } \} \\
& X/F.X \sqsupseteq R.X \\
\equiv & \quad \{ \text{ left factors } \} \\
& X \;\sqsupseteq\; R.X \circ F.X \\
\equiv & \quad \{ \text{ assumption } \} \\
& true
\end{array}
$$

We make now the following definition:

Definition 84: spec X is *right F–recursive* $\quad \widehat{=} \quad X = X/F.X \circ F.X$

We conclude this section with the observation that a per being F–inductive is equivalent to it being right F–recursive. Being precise, we have:

Lemma 85: $per.A \;\;\Rightarrow\;\; (A = A/F.A \circ F.A \;\;\equiv\;\; A \circ F.A = A)$

Proof "$\Rightarrow$"

$$
\begin{array}{ll}
& A \circ F.A \\
= & \quad \{ \text{ assumption } \} \\
& A/F.A \circ F.A \circ F.A \\
= & \quad \{ \text{ distribution } \} \\
& A/F.A \;\circ\; F.(A \circ A) \\
= & \quad \{ \text{ per.}A \text{ } \} \\
& A/F.A \circ F.A \\
= & \quad \{ \text{ assumption } \} \\
& A
\end{array}
$$

Proof "$\Leftarrow$"

$$
\begin{array}{ll}
& A \\
\sqsupseteq & \quad \{ \text{ cancellation } \} \\
& A/F.A \circ F.A \\
= & \quad \{ \text{ assumption } \} \\
& (A \circ F.A)\,/\,F.A \;\circ\; F.A \\
\sqsupseteq & \quad \{ \text{ cancellation } \} \\
& A \circ F.A \\
= & \quad \{ \text{ assumption } \} \\
& A
\end{array}
$$

4.5 Inductive types with laws

Our basic method of making inductive types is to take the least or greatest fixpoint of some relator F. We present in this section another method. Given an F–inductive type B, imposing laws on the constructors of B gives another F–inductive type. Given such a *base type* B, and a set $\mathcal{S}$ of spec pairs, we seek the least per C under the spec ordering "$\sqsubseteq$" that satisfies the following 3 properties:

$(P1)\ \ C \lhd| B$

$(P2)\ \ C \lhd| F.C$

$(P3)\ \ C \circ f = C \circ g$, for all $(f, g) \in \mathcal{S}$

$P1$ says that C must be an equivalence relation on B; we are not allowed to discard or create new elements, on coalesce; $P2$ is explained in the next paragraph; $P3$ says that elements in B identified by f and g are coalesced in C. (Note that we do not in fact require that the specs in $\mathcal{S}$ be difunctional, but in our work to date no non–difunctionals have occurred.) Taking the least spec C satisfying $P1$–$P3$ ensures that we coalesce precisely those elements of B as demanded by the laws; $P3$ on its own does not prevent us coalescing extra elements.

Let us now explain $P2$ above. Requiring that $C \lhd F.C$ is natural, but why the stronger $C \lhd| F.C$? Consider the natural numbers as least fixpoint μF of the relator $F.X = \top\!\!\top + X$. Using the results of the previous section, we have constructors $zero \in \mu F \leftarrow \top\!\!\top$ and $succ \in \mu F \leftarrow \mu F$. Noting that $f \in A \leftarrow \top\!\!\top \Rightarrow f \circ f^\cup \in A$, we see that the number 0 is represented by $zero \circ zero^\cup$, the number 1 by $(succ \circ zero) \circ (succ \circ zero)^\cup$, etc. Choose for B the subtype of μF with elements 0 and 1 only; choose for $\mathcal{S}$ the singleton set $\{(zero, succ \circ zero)\}$. We observe now that the only per C satisfying $P1$ and $P3$ is that with 0 and 1 coalesced to form a single element. But this type does not satisfy $P2$; it is not F–inductive. In the proofs, we find that assuming $B \lhd| F.B$ rather than the weaker $B \lhd F.B$ is sufficient to ensure that a per C satisfying $P1$–$P3$ exists. Under the assumption that $B \lhd| F.B$, it is natural then that we should require $C \lhd| F.C$, thereby allowing the new type C itself to be used as a base type in imposing further laws on B.

Assumptions: for all $(f, g) \in \mathcal{S}$

$(a)\ \ B \lhd| F.B$

$(b)\ \ B \sqsupseteq f \circ f^\cup$

$(c)\ \ B \sqsupseteq g \circ g^\cup$

$(d)\ \ f> = g>$

Assumption (a) is explained above; it is important to note that $B \lhd| F.B$ means that laws can only be applied to unrestricted types. Assumptions *b*–*d* are used in the proof that $P1$–$P3$ have a least solution for C; in practice, *b*–*d* are satisfied by choosing f and g as total difunctionals of type $B \leftarrow A$ for some per A.

It is precisely the equations $C \circ f = C \circ g$ in $P3$ above that we call *laws*. Let us give a simple example. Consider the relator defined by $F.X = \top\!\!\top + X \times X$. We may view the least fixpoint μF as a type of *shapes* of binary trees, with constructors $[]_{\mu F} \in \mu F \leftarrow \top\!\top$ and $+\!\!+_{\mu F} \in \mu F \leftarrow \mu F \times \mu F$. From the type μF we can make a type of *mobiles* (delicately balanced sculptures hung from the ceiling) by imposing the law that the "$+\!\!+$" constructor be symmetric, and the restriction that "$+\!\!+$" only be applicable to subcomponents whose weight differs by at most one. We consider here only the law; the restriction is treated in the next section.

We seek an F–inductive per C, with constructors $[]_C$ and $+\!\!+_C$, satisfying the law $+\!\!+_C \circ \bowtie \; = +\!\!+_C$, where $\bowtie \;\; \widehat{=} \; \gg \vartriangle \ll$ (pronounced "bow tie") swaps the components of a pair. The law says that the "$+\!\!+$" constructor is required to be symmetric. The task now is to cast the law in the form of $P3$ above, i.e. $C \circ f = C \circ g$ for some specs f and g that are independent of C. The following calculation shows that $f := +\!\!+_{\mu F} \circ \bowtie$ and $g := +\!\!+_{\mu F}$ work. This concludes our example.

$$
\begin{array}{cl}
 & +\!\!+_C \circ \bowtie \; = +\!\!+_C \\
\equiv & \qquad \{ \ \text{def } +\!\!+_C \ \} \\
 & C \circ \hookleftarrow \circ \bowtie \; = C \circ \hookleftarrow \\
\equiv & \qquad \{ \ C \lhd \mu F \ \} \\
 & C \circ \mu F \circ \hookleftarrow \circ \bowtie \; = C \circ \mu F \circ \hookleftarrow \\
\equiv & \qquad \{ \ \text{def } +\!\!+_{\mu F} \ \} \\
 & C \circ +\!\!+_{\mu F} \circ \bowtie \; = C \circ +\!\!+_{\mu F} \\
\equiv & \qquad \{ \ f := +\!\!+_{\mu F} \circ \bowtie, \; g := +\!\!+_{\mu F} \ \} \\
 & C \circ f = C \circ g
\end{array}
$$

Let us write $P.C$ to mean that per C satisfies $P1$–$P3$. Looking at these three predicates, it is not clear that *any* per C exists for which $P.C$ holds, never mind a least. In particular, $P1$–$P3$ are not in a form to which the Knaster–Tarski fixpoint theorem may be applied. Consider however the following inclusions in X:

$(Q1)$ $X \sqsupseteq X \circ X$

$(Q2)$ $X \sqsupseteq F.X$

$(Q3)$ $X \sqsupseteq B$

$(Q4)$ $X \sqsupseteq \bigsqcup (f, g : (f, g) \in \mathcal{S} : f \circ g^{\cup})$

$(Q5)$ $X \sqsupseteq \bigsqcup (f, g : (f, g) \in \mathcal{S} : g \circ f^{\cup})$

It is clear that the right side of $Q1$–$Q5$ are monotonic functions in X; the existence of a least X satisfying $Q1$–$Q5$, call it $B /\!\!/ \mathcal{S}$, follows then from the Knaster–Tarski theorem. Let us write $X \sqsupseteq Q.X$ to means that X satisfies $Q1$–$Q5$. (Formally, we may define Q as the "$\sqcup$" of the right sides of these five inclusions.)

Theorem 86: $B /\!\!/ \mathcal{S}$ satisfies $P1$–$P3$, and is the least spec doing so:

$$P.(B /\!\!/ \mathcal{S}) \;\; \wedge \;\; \forall (X :: X \sqsupseteq B /\!\!/ \mathcal{S} \;\; \Leftarrow \;\; P.X)$$

We make a few remarks:

- We are not asserting that the complete set of specs which satisfy $P1$–$P3$ and $Q1$–$Q5$ coincide, only that the least such specs do. (We do however find during the proof of the second conjunct above that the second set subsumes the first.)
- We have no justification for the choice of $Q1$–$Q5$ other than the above theorem. We can however offer some remarks about each clause: $Q1$ expresses transitivity of X; $Q2$ ensures totality, $Q3$ ensures $X \lhd\!\mid F.X$; $Q4$ and $Q5$ ensure that elements in B identified by the f and g's are coalesced in X.
- The $B /\!\!/ \mathcal{S}$ notation is chosen because, just as with the division operator in arithmetic, the $/\!\!/$ operator is found to be anti–monotonic in its numerator, and monotonic in its denominator. Being precise, we have:

$$B /\!\!/ \mathcal{S} \lhd\!\mid C /\!\!/ \mathcal{S} \;\Leftarrow\; B \lhd\!\mid C$$

$$B /\!\!/ \mathcal{S} \lhd\!\mid B /\!\!/ \mathcal{T} \;\Leftarrow\; S \supseteq T$$

The first step in proving the above theorem is to show that $P.(B /\!\!/ \mathcal{S})$ holds:

(1) $per.(B /\!\!/ \mathcal{S})$
(2) $B /\!\!/ \mathcal{S} \lhd\!\mid B$
(3) $B /\!\!/ \mathcal{S} \lhd\!\mid F.(B /\!\!/ \mathcal{S})$
(4) $B /\!\!/ \mathcal{S} \circ f = B /\!\!/ \mathcal{S} \circ g$, for all $(f, g) \in \mathcal{S}$

Proof 1: Transitivity of $B /\!\!/ \mathcal{S}$ is immediate from the the first clause of the definition of $Q.X$. For symmetry of $B /\!\!/ \mathcal{S}$, we calculate as follows:

$$
\begin{array}{ll}
 & (B /\!\!/ \mathcal{S})^{\cup} \sqsupseteq B /\!\!/ \mathcal{S} \\
\Leftarrow & \qquad \{ \text{ fixpoint induction } \} \\
 & (B /\!\!/ \mathcal{S})^{\cup} \sqsupseteq Q.((B /\!\!/ \mathcal{S})^{\cup}) \\
\equiv & \qquad \{ \text{ lemma: } Q.(R^{\cup}) = (Q.R)^{\cup} \} \\
 & (B /\!\!/ \mathcal{S})^{\cup} \sqsupseteq (Q.B /\!\!/ \mathcal{S})^{\cup} \\
\equiv & \qquad \{ \text{ def } B /\!\!/ \mathcal{S} \} \\
 & true
\end{array}
$$

Proof 2

$$
\begin{array}{ll}
 & B /\!\!/ \mathcal{S} \lhd\!\mid B \\
\equiv & \qquad \{ \text{ def } \lhd\!\mid \} \\
 & B /\!\!/ \mathcal{S} \circ B \;=\; B /\!\!/ \mathcal{S} \;\wedge\; uQ \sqsupseteq B \\
\equiv & \qquad \{ \; B /\!\!/ \mathcal{S} \text{ solves } X \sqsupseteq Q.X \; \} \\
 & B /\!\!/ \mathcal{S} \circ B \;=\; B /\!\!/ \mathcal{S} \\
\equiv & \qquad \{ \text{ antisymmetry } \} \\
 & B /\!\!/ \mathcal{S} \circ B \;\sqsupseteq\; B /\!\!/ \mathcal{S} \;\;\wedge\;\; B /\!\!/ \mathcal{S} \;\sqsupseteq\; B /\!\!/ \mathcal{S} \circ B \\
\equiv & \qquad \{ \; B /\!\!/ \mathcal{S} \sqsupseteq B /\!\!/ \mathcal{S} \circ B /\!\!/ \mathcal{S} \sqsupseteq B /\!\!/ \mathcal{S} \circ B \; \} \\
 & B /\!\!/ \mathcal{S} \circ B \;\sqsupseteq\; B /\!\!/ \mathcal{S}
\end{array}
$$

$$
\begin{array}{ll}
\Leftarrow & \quad \{ \text{ fixpoint induction } \} \\
& (2.1)\ \ B/\!\!/\mathcal{S} \circ B \sqsupseteq B/\!\!/\mathcal{S} \circ B \circ B/\!\!/\mathcal{S} \circ B \\
& (2.2)\ \ B/\!\!/\mathcal{S} \circ B \ \sqsupseteq \ F.(B/\!\!/\mathcal{S} \circ B) \\
& (2.3)\ \ B/\!\!/\mathcal{S} \circ B \ \sqsupseteq \ B \sqcup f \circ g^{\cup} \sqcup g \circ f^{\cup}, \text{ for all } (f,g) \in \mathcal{S}
\end{array}
$$

Proof 2.1

$$
\begin{array}{ll}
& B/\!\!/\mathcal{S} \circ B \\
= & \quad \{ \text{ per.} B/\!\!/\mathcal{S} \ \} \\
& B/\!\!/\mathcal{S} \circ B/\!\!/\mathcal{S} \circ B/\!\!/\mathcal{S} \circ B \\
\sqsupseteq & \quad \{ \ B/\!\!/\mathcal{S} \sqsupseteq B \ \} \\
& B/\!\!/\mathcal{S} \circ B \circ B/\!\!/\mathcal{S} \circ B
\end{array}
$$

Proof 2.2

$$
\begin{array}{ll}
& B/\!\!/\mathcal{S} \circ B \\
\sqsupseteq & \quad \{ \ B \sqsupseteq F.B \ \} \\
& B/\!\!/\mathcal{S} \circ F.B \\
\sqsupseteq & \quad \{ \ B/\!\!/\mathcal{S} \sqsupseteq F.(B/\!\!/\mathcal{S}) \ \} \\
& F.(B/\!\!/\mathcal{S}) \circ F.B \\
= & \quad \{ \text{ relators } \} \\
& F.(B/\!\!/\mathcal{S} \circ B)
\end{array}
$$

Proof 2.3: for all $(f,g) \in \mathcal{S}$

$$
\begin{array}{ll}
& B/\!\!/\mathcal{S} \circ B \\
\sqsupseteq & \quad \{ \ B/\!\!/\mathcal{S} \sqsupseteq B \sqcup f \circ g^{\cup} \sqcup g \circ f^{\cup} \ \} \\
& (B \sqcup f \circ g^{\cup} \sqcup g \circ f^{\cup}) \circ B \\
= & \quad \{ \text{ distribution } \} \\
& (B \circ B) \sqcup (f \circ g^{\cup} \circ B) \sqcup (g \circ f^{\cup} \circ B) \\
\sqsupseteq & \quad \{ \ B \sqsupseteq g \circ g^{\cup},\ B \sqsupseteq f \circ f^{\cup} \ \} \\
& (B \circ B) \sqcup (f \circ g^{\cup} \circ g \circ g^{\cup}) \sqcup (g \circ f^{\cup} \circ f \circ f^{\cup}) \\
\sqsupseteq & \quad \{ \ B = B \circ B,\ R \circ R^{\cup} \circ R \sqsupseteq R \ \} \\
& B \sqcup f \circ g^{\cup} \sqcup g \circ f^{\cup}
\end{array}
$$

Proof 3

$$
\begin{array}{ll}
& B/\!\!/\mathcal{S} \lhd\!\mid F.(B/\!\!/\mathcal{S}) \\
\equiv & \quad \{ \text{ def } \lhd\!\mid \ \} \\
& B/\!\!/\mathcal{S} \circ F.(B/\!\!/\mathcal{S}) = B/\!\!/\mathcal{S} \quad \wedge \quad B/\!\!/\mathcal{S} \sqsupseteq F.(B/\!\!/\mathcal{S}) \\
\equiv & \quad \{ \ B/\!\!/\mathcal{S} \text{ solves } X \sqsupseteq Q.X \ \} \\
& B/\!\!/\mathcal{S} \circ F.(B/\!\!/\mathcal{S}) = B/\!\!/\mathcal{S} \\
\equiv & \quad \{ \text{ antisymmetry } \} \\
& B/\!\!/\mathcal{S} \circ F \sqsupseteq B/\!\!/\mathcal{S} \quad \wedge \quad B/\!\!/\mathcal{S} \sqsupseteq B/\!\!/\mathcal{S} \circ F.(B/\!\!/\mathcal{S})
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \quad \{\ B/\!\!/\mathcal{S} \sqsupseteq B/\!\!/\mathcal{S} \circ B/\!\!/\mathcal{S} \sqsupseteq B/\!\!/\mathcal{S} \circ F.(B/\!\!/\mathcal{S})\ \} \\
& B/\!\!/\mathcal{S} \circ F.(B/\!\!/\mathcal{S}) \sqsupseteq B/\!\!/\mathcal{S} \\
\Leftarrow & \quad \{\ B/\!\!/\mathcal{S} \text{ solves } X \sqsupseteq Q.X\ \} \\
& B/\!\!/\mathcal{S} \circ F.B \sqsupseteq B/\!\!/\mathcal{S} \\
\equiv & \quad \{\ B/\!\!/\mathcal{S} = B/\!\!/\mathcal{S} \circ B\ \} \\
& B/\!\!/\mathcal{S} \circ B \circ F.B \sqsupseteq B/\!\!/\mathcal{S} \\
\equiv & \quad \{\ B \lhd\!| \ F.B\ \} \\
& B/\!\!/\mathcal{S} \circ B \sqsupseteq B/\!\!/\mathcal{S} \\
\equiv & \quad \{\ B/\!\!/\mathcal{S} = B/\!\!/\mathcal{S} \circ B\ \} \\
& true
\end{array}
$$

Proof 4: We show only one inclusion. For all $(f, g) \in \mathcal{S}$:

$$
\begin{array}{ll}
& B/\!\!/\mathcal{S} \circ f \\
= & \quad \{\ \text{per}.B/\!\!/\mathcal{S}\ \} \\
& B/\!\!/\mathcal{S} \circ B/\!\!/\mathcal{S} \circ f \\
\sqsupseteq & \quad \{\ B/\!\!/\mathcal{S} \sqsupseteq g \circ f\cup\ \} \\
& B/\!\!/\mathcal{S} \circ g \circ f\cup \circ f \\
\sqsupseteq & \quad \{\ \text{domains}\ \} \\
& B/\!\!/\mathcal{S} \circ g \circ f> \\
= & \quad \{\ f> = g>\ \} \\
& B/\!\!/\mathcal{S} \circ g \circ g> \\
= & \quad \{\ \text{domains}\ \} \\
& B/\!\!/\mathcal{S} \circ g
\end{array}
$$

This completes the proof that $P.(B/\!\!/\mathcal{S})$ holds. We are now required to show that $B/\!\!/\mathcal{S}$ is the least spec satisfying P. We proceed as follows:

$$
\begin{array}{ll}
& \forall (X :: X \sqsupseteq B/\!\!/\mathcal{S} \Leftarrow P.X) \\
\Leftarrow & \quad \{\ \text{fixpoint induction}\ \} \\
& \forall (X :: X \sqsupseteq Q.X \Leftarrow P.X)
\end{array}
$$

Assuming $P.X$, we are thus required to show

(5) $X \sqsupseteq X \circ X$

(6) $X \sqsupseteq F.X$

(7) $X \sqsupseteq B$

(8) $X \sqsupseteq \sqcup (f, g : (f, g) \in \mathcal{S} : f \circ g\cup)$

(9) $X \sqsupseteq \sqcup (f, g : (f, g) \in \mathcal{S} : g \circ f\cup)$

Inclusions 5–7 follow immediately from the assumption that $P.X$ holds. We find moreover that inclusions 8 and 9 are equivalent:

$$
\begin{array}{ll}
& X \sqsupseteq f \circ g^{\cup} \\
\equiv & \quad \{ \text{ reverse } \} \\
& X^{\cup} \sqsupseteq (f \circ g^{\cup})^{\cup} \\
\equiv & \quad \{ \text{ per}.X, \text{ reverse } \} \\
& X \sqsupseteq g \circ f^{\cup}
\end{array}
$$

Showing for all $(f,g) \in \mathcal{S}$ that $X \sqsupseteq f \circ g^{\cup}$ thus completes our proof:

$$
\begin{array}{ll}
& X \\
= & \quad \{ X \lhd B \} \\
& X \circ B \\
\sqsupseteq & \quad \{ B \sqsupseteq g \circ g^{\cup} \} \\
& X \circ g \circ g^{\cup} \\
= & \quad \{ X \circ g = X \circ f \} \\
& X \circ f \circ g^{\cup} \\
\sqsupseteq & \quad \{ X \sqsupseteq B \} \\
& B \circ f \circ g^{\cup} \\
\sqsupseteq & \quad \{ B \sqsupseteq f \circ f^{\cup} \} \\
& f \circ f^{\cup} \circ f \circ g^{\cup} \\
\sqsupseteq & \quad \{ \text{ triple rule } \} \\
& f \circ g^{\cup}
\end{array}
$$

We conclude this section by verifying two properties we expect of types with laws. Firstly, that laws can be imposed independently of one another:

Lemma 87: $B /\!\!/ (\mathcal{S} \cup \mathcal{T}) = (B /\!\!/ \mathcal{S}) /\!\!/ \mathcal{T}$

We omit the proof of this lemma; it is a little messy, but not difficult.

Given some relator F, every F–catamorphism $(\![R]\!)$ is well–defined on the least fixpoint type μF, in the sense that $(\![R]\!) \circ \mu F = (\![R]\!)$. Generalising from catamorphisms to right recursive specs, and from least fixpoint types to types with laws, the lemma below gives precise conditions under which a right recursive spec X is well defined on $B /\!\!/ \mathcal{S}$. The first conjunct says that X must be well–defined on the base type B, the second that X must respect the laws represented in $\mathcal{S}$.

Lemma 88: For a right F–recursive spec X,

$$
\begin{array}{ll}
& X \circ B /\!\!/ \mathcal{S} = X \\
\equiv & \\
& X \circ B = X \quad \wedge \quad \forall\,(f,g : (f,g) \in S : X \circ f = X \circ g)
\end{array}
$$

Proof "$\Rightarrow .1$"

$$
\begin{array}{ll}
 & X \circ B \\
= & \quad \{\ X \circ B /\!\!/ \mathcal{S} = X\ \} \\
 & X \circ B /\!\!/ \mathcal{S} \circ B \\
= & \quad \{\ B /\!\!/ \mathcal{S} \lhd\!\mid B\ \} \\
 & X \circ B /\!\!/ \mathcal{S} \\
= & \quad \{\ X \circ B /\!\!/ \mathcal{S} = X\ \} \\
 & X
\end{array}
$$

Proof "$\Rightarrow .2$"

$$
\begin{array}{ll}
 & X \circ f \\
= & \quad \{\ X \circ B /\!\!/ \mathcal{S} = X\ \} \\
 & X \circ B /\!\!/ \mathcal{S} \circ f \\
= & \quad \{\ B /\!\!/ \mathcal{S} \circ f = B /\!\!/ \mathcal{S} \circ g\ \} \\
 & X \circ B /\!\!/ \mathcal{S} \circ g \\
= & \quad \{\ X \circ B /\!\!/ \mathcal{S} = X\ \} \\
 & X \circ g
\end{array}
$$

Proof "$\Leftarrow$"

$$
\begin{array}{ll}
 & X \circ B /\!\!/ \mathcal{S} = X \\
\equiv & \quad \{\ X \circ B /\!\!/ \mathcal{S} \sqsupseteq X \circ B = X\ \} \\
 & X \sqsupseteq X \circ B /\!\!/ \mathcal{S} \\
\equiv & \quad \{\ \text{factors}\ \} \\
 & X \setminus X \sqsupseteq B /\!\!/ \mathcal{S} \\
\Leftarrow & \quad \{\ \text{fixpoint induction}\ \} \\
 & (1)\ \ X \setminus X \ \sqsupseteq\ X \setminus X \circ X \setminus X \\
 & (2)\ \ X \setminus X \ \sqsupseteq\ F.(X \setminus X) \\
 & (3)\ \ X \setminus X \ \sqsupseteq\ B \ \sqcup\ f \circ g^{\cup} \ \sqcup\ g \circ f^{\cup}
\end{array}
$$

Proof 1

$$
\begin{array}{ll}
 & X \backslash X \sqsupseteq X \backslash X \circ X \backslash X \\
\equiv & \quad \{\ \text{factors}\ \} \\
 & X \sqsupseteq X \circ X \backslash X \circ X \backslash X \\
\Leftarrow & \quad \{\ \text{cancellation}\ \} \\
 & X \sqsupseteq X \circ X \backslash X \\
\equiv & \quad \{\ \text{again}\ \} \\
 & true
\end{array}
$$

Proof 2

$$
\begin{array}{ll}
 & X \setminus X \ \sqsupseteq\ F.(X \setminus X) \\
\equiv & \quad \{\ \text{factors}\ \}
\end{array}
$$

$$
\begin{array}{ll}
 & X \sqsupseteq X \circ F.(X \setminus X) \\
\equiv & \qquad \{ \ X \text{ is right } F\text{–recursive } \} \\
 & X \sqsupseteq X \,/\, F.X \circ F.X \circ F.(X \setminus X) \\
\equiv & \qquad \{ \text{ relators } \} \\
 & X \sqsupseteq X \,/\, F.X \circ F.(X \circ X \setminus X) \\
\equiv & \qquad \{ \text{ factors, relators } \} \\
 & X \sqsupseteq X \,/\, F.X \circ F.X \\
\equiv & \qquad \{ \text{ factors } \} \\
 & true
\end{array}
$$

Proof 3

$$
\begin{array}{ll}
 & X \backslash X \sqsupseteq B \sqcup f \circ g^{\cup} \sqcup g \circ f^{\cup} \\
\equiv & \qquad \{ \text{ factors } \} \\
 & X \sqsupseteq X \circ (B \sqcup f \circ g^{\cup} \sqcup g \circ f^{\cup}) \\
\equiv & \qquad \{ \text{ distribution } \} \\
 & X \sqsupseteq X \circ B \sqcup X \circ f \circ g^{\cup} \sqcup X \circ g \circ f^{\cup} \\
\equiv & \qquad \{ \ X \circ f = X \circ g \ \} \\
 & X \sqsupseteq X \circ B \sqcup X \circ g \circ g^{\cup} \sqcup X \circ f \circ f^{\cup} \\
\Leftarrow & \qquad \{ \ B \sqsupseteq g \circ g^{\cup}, B \sqsupseteq f \circ f^{\cup} \ \} \\
 & X \sqsupseteq X \circ B \\
\equiv & \qquad \{ \ X = X \circ B \ \} \\
 & true
\end{array}
$$

4.6 Inductive types with restrictions

In the last section it was shown how to make a new type by imposing laws on the constructors of an existing type. In this section, we show how to make a new type by imposing *restrictions* on the constructors. Just as for types with laws, we begin with the assumption that our base–type B be inductive with respect to some relator F; unlike with laws, we do not require that the constructors be total on B.

Assumption: $B \lhd F.B$

Imposing restrictions on the constructors for B gives rise to a *subtype*, i.e. some per A satisfying $A \subseteq B$. So that it may be used as a base–type in imposing further restrictions on B, we require that the new type A be itself F–inductive. How do we make such a subtype? The ordering operator "$\subseteq$" tells us only how to recognise subtypes; the lemma below gives a practical method for making subtypes:

Lemma 89: (subtype lemma)

$$per.B \ \wedge \ R \circ B = R \quad \Rightarrow \quad per.(B \circ R{>}) \ \wedge \ (B \circ R{>}) \subseteq B$$

(Recall that $R \circ B = R$ says that R on its right–side respects the elements of B.) Expanding for "$\subseteq$" above, $(B \circ R{>}) \subseteq B$ follows trivially from the assumptions and simple domain properties; we prove the remaining part, $per.(B \circ R{>})$, in two steps:

Proof (symmetry)

$$
\begin{array}{ll}
 & B \ \circ \ R{>} \\
= & \qquad \{ \ \text{def} > \ \} \\
 & B \circ (I \ \sqcap \ \top\!\top \circ R) \\
= & \qquad \{ \ \text{distribution} \ \} \\
 & (B \circ I) \sqcap (\top\!\top \circ R) \\
= & \qquad \{ \ \text{identity} \ \} \\
 & (I \circ B) \sqcap (\top\!\top \circ R) \\
= & \qquad \{ \ R \circ B = R \ \} \\
 & (I \circ B) \sqcap (\top\!\top \circ R \circ B) \\
= & \qquad \{ \ \text{distribution} \ \} \\
 & (I \ \sqcap \ \top\!\top \circ R) \circ B \\
= & \qquad \{ \ \text{def} > \ \} \\
 & R{>} \ \circ \ B \\
= & \qquad \{ \ \text{per}.B \ \} \\
 & (R{>} \ \circ \ B)\cup
\end{array}
$$

Proof (transitivity)

$$
\begin{array}{ll}
 & B \ \circ \ R{>} \\
= & \qquad \{ \ \text{per}.B, \text{domains} \ \} \\
 & B \ \circ \ B{>} \ \circ \ B \ \circ \ R{>} \\
\sqsupseteq & \qquad \{ \ R \circ B = R \ \} \\
 & B \ \circ \ R{>} \ \circ \ B \ \circ \ R{>}
\end{array}
$$

In general, the subtype $B \circ R{>} \subseteq B$ is not F–inductive (recall that inductivity is required to allow the subtype itself to be used as a base–type in imposing further restrictions on B). We have however a precise condition for inductivity:

Lemma 90: Given $per.B$ and $R \circ B = R$,

$$(B \circ R{>}) \text{ is } F\text{–}inductive \quad \equiv \quad F.R{>} \ \sqsupseteq \ R{>}$$

Proof "$\Rightarrow$"

$$
\begin{array}{ll}
 & F.R{>} \\
\sqsupseteq & \qquad \{ \ \text{per}.X \ \Rightarrow \ X \sqsupseteq X{>} \ \} \\
 & (F.R{>}){>} \\
\sqsupseteq & \qquad \{ \ \text{domains} \ \} \\
 & (B \ \circ \ R{>} \ \circ \ F.B \ \circ \ F.R{>}){>} \\
= & \qquad \{ \ \text{relators} \ \}
\end{array}
$$

$$
\begin{array}{ll}
 & (B \circ R{>} \circ F.(B \circ R{>}))> \\
= & \quad \{\ B \circ R{>} \text{ is inductive } \} \\
 & (B \circ R{>})> \\
= & \quad \{\ \text{domains } \} \\
 & (B{>} \circ R{>})> \\
= & \quad \{\ R \circ B = R, \text{ domains } \} \\
 & (R{>})> \\
= & \quad \{\ \text{domains } \} \\
 & R{>}
\end{array}
$$

Proof "$\Leftarrow$"

$$
\begin{array}{ll}
 & \mathit{inductive}.(B \circ R{>}) \\
\equiv & \quad \{\ \text{def inductive } \} \\
 & B \circ R{>} \circ F.(B \circ R{>}) \ = \ B \circ R{>}
\end{array}
$$

We proceed to verify the equality above by mutual inclusion:

Proof "$\sqsupseteq$"

$$
\begin{array}{ll}
 & B \circ R{>} \circ F.(B \circ R{>}) \\
= & \quad \{\ \text{relators } \} \\
 & B \circ R{>} \circ \circ F.R{>} \\
= & \quad \{\ B \circ R{>} \text{ is a per } \} \\
 & (B \circ R{>})^{\cup} \circ, \circ F.R{>} \\
= & \quad \{\ \text{reverse, pers } \} \\
 & R{>} \circ B \circ \circ F.R{>} \\
= & \quad \{\ \text{inductive}.B \ \} \\
 & R{>} \circ B \circ F.R{>} \\
\sqsupseteq & \quad \{\ F.(R{>}) \sqsupseteq R{>} \ \} \\
 & R{>} \circ B \circ R{>} \\
= & \quad \{\ B \circ R{>} \text{ is a per } \} \\
 & B \circ R{>} \circ R{>} \\
= & \quad \{\ \text{per}.R{>} \ \} \\
 & B \circ R{>}
\end{array}
$$

Proof "$\sqsubseteq$"

$$
\begin{array}{ll}
 & B \circ R{>} \\
= & \quad \{\ B \circ R{>} \text{ is a per } \} \\
 & (B \circ R{>})^{\cup} \\
= & \quad \{\ \text{per}.B, \text{ per}.R{>} \ \} \\
 & R{>} \circ B \\
= & \quad \{\ \text{inductive}.B \ \} \\
 & R{>} \circ B \circ F.B
\end{array}
$$

$$
\begin{array}{ll}
\sqsupseteq & \{\ B \sqsupseteq B \circ R{>},\ \text{relators}\ \} \\
& R{>} \circ B \circ F.(B \circ R{>}) \\
= & \{\ B \circ R{>}\ \text{is a per}\ \} \\
& B \circ R{>} \circ F.(B \circ R{>})
\end{array}
$$

Let us summarise our construction of subtypes:

Lemma 91: Given $R \circ B = R$ and $F.R{>} \sqsupseteq R{>}$,

$(B \circ R{>}) \subseteq B$ is F*–inductive*

In practice, the precondition $F.R{>} \sqsupseteq R{>}$ is satisfied by choosing R as a right–recursive spec, e.g. a catamorphism; that this is sufficient is proved below.

$$
\begin{array}{ll}
& F.R{>} \\
\sqsupseteq & \{\ \text{relators, domains}\ \} \\
& (F.R){>} \\
\sqsupseteq & \{\ \text{domains}\ \} \\
& (R/F.R \circ F.R){>} \\
= & \{\ \text{recursive}.R\ \} \\
& R{>}
\end{array}
$$

To clarify the construction of subtypes, let us return to the "mobiles" example of the last section. Given the relator $F.X = \top\!\!\top + X{\times}X$, we view μF as a type of shapes of binary trees, with constructors $[]_{\mu F}$ and $+\!\!+_{\mu F}$. The type of *mobiles* is gained from μF by imposing the law that "$+\!\!+$" be symmetric, and the restriction that "$+\!\!+$" only be applicable to subcomponents whose weight differs by at most one unit. In the last section we showed that imposing the law gives the type $\mu F /\!\!/ \{(+\!\!+_{\mu F} \circ \bowtie, +\!\!+_{\mu F})\}$.

Let us now consider the restriction. The *weight* of a tree is given by the catamorphism $(\![1 \triangledown +]\!)$. Define a partial operator "@" by $x \mathbin{@} y = x+y,\ \textit{if}\ |x-y| \leq 1$; the catamorphism $(\![1 \triangledown @]\!)$ is then itself partial, giving the weight only for balanced trees. Using the results of this section, the type of mobiles is given by

$$\mu F /\!\!/ \{(+\!\!+_{\mu F} \circ \bowtie, +\!\!+_{\mu F})\} \circ (\![1 \triangledown @]\!){>}$$

Chapter 5

Summary and future directions

Backhouse et al in [3] present a calculational theory of programming, with binary relations as programs and partial identity relations (monotypes) as types. We in this article work within the same framework, but adopt a more general notion of type: partial equivalence relations (pers). Working with pers allows us to handle types with laws. For example, recalling that $\mathcal{S}^2$ abbreviates the full relation $\mathcal{S}\times\mathcal{S}$, imposing the law $a = b$ on the monotype $\{a\}^2\cup\{b\}^2\cup\{c\}^2$ gives the per $\{a,b\}^2\cup\{c\}^2$. A quite different treatment of types with laws in the *F–algebra* style of type definition has recently been given by Fokkinga [15]; the relationship with our approach is clearly an interesting topic for study.

Working with monotypes, the functional specs (imps) are precisely those f satisfying $I \sqsupseteq f \circ f^{\cup}$. With pers, we found the functional specs to be precisely those f satisfying $f = f \circ f^{\cup} \circ f$, the so–called difunctionals. Moreover, we presented three alternative ways to look at difunctionals: as the union of disjoint bundles, co–imp/imp factorisable specs, and invertible specs. Pers are being used in many areas in computing science (e.g. modelling the λ–calculus and in analysing lazy functional programs.) It seems likely then that difunctionals, not a familiar concept to most computing scientists, have many other interesting applications.

Difunctionals generalise imps; another such generalisation is *causal relations* [16], throwing away the constraint that inputs must always appear on the right–side of a functional relation, and outputs on the left–side. Causal relations are important in the derivation of programs that can be implemented in hardware, being "implementable relations" in Jones and Sheeran's calculational approach to circuit design known as Ruby [17]. A simulator for causal Ruby programs has been produced [18]. We are experimenting with implementing difunctional relations, with a view to producing a more general simulator based upon causal relations as a generalisation of difunctionals rather than imps. Since pers are now adopted as types in Ruby [19, 20], such a simulator should accept Ruby programs earlier in the design process, and in some cases even the initial specification itself.

Our article is mainly theoretical. The natural question is a practical one: what benefits do working with relations as programs and pers as types bring in calculating programs from specifications ? The answer we leave to future articles.

Acknowledgements

Thanks are due to Roland Backhouse, Erik Meijer and Jaap van der Woude for their useful comments and suggestions. The first author also thanks Backhouse's relational type theory group for their hospitality during two visits to Eindhoven.

Bibliography

[1] Richard Bird. *Lectures on constructive functional programming.* Oxford University 1988. (PRG–69)

[2] Grant Malcolm. *Algebraic data types and program transformation.* Ph.D. thesis, Groningen University, 1990.

[3] Roland Backhouse, Ed Voermans and Jaap van der Woude. *A relational theory of datatypes.* Appears in [21].

[4] Graham Hutton and Ed Voermans. *A calculational theory of pers as types.* Appears in [21].

[5] Graham Hutton and Ed Voermans. *Making functionality more general.* Proc. Glasgow Workshop on Functional Programming, Skye 1991. To appear in Springer Workshops in Computing.

[6] E.W. Dijkstra and W.H.J. Feijen. *A Method of Programming.* Addison–Wesley, 1988.

[7] A.J.M. van Gasteren. *On the shape of mathematical arguments.* LNCS 445, Springer–Verlag, Berlin, 1990.

[8] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer–Verlag, Berlin, 1990.

[9] C.A.R. Hoare and Jifeng He. *The weakest prespecification.* Fundamenta Informaticae, 9:51–84, 217–252, 1986.

[10] Frans Rietman. *A note on extensionality.* Appears in [21].

[11] Ed Voermans. *The equivalence domain.* Appears in [21].

[12] J. Riguet. *Relations Binaires, Fermetures, Correspondances de Galois.* Bulletin de la Societé mathématique de France. Volume 76, 1948.

[13] A. Jaoua, A. Mili, N. Boudriga, and J.L. Durieux. *Regularity of relations: A measure of uniformity.* Theoretical Computer Science 79, 323–339, 1991.

[14] Maarten Fokkinga and Erik Meijer. *Program calculation properties of continuous algebras.* Appears in [21].

[15] Maarten Fokkinga. *Datatype laws without signatures.* Appears in [21].

[16] Graham Hutton. *Functional programming with relations.* Proc. Third Glasgow Workshop on functional programming (ed. Kehler Holst, Hutton and Peyton Jones), Ullapool 1990, Springer Workshops in Computing.

[17] Geraint Jones and Mary Sheeran. *Relations + higher–order functions = hardware descriptions.* Proc. IEEE Comp Euro 87: VLSI and Computers, Hamburg, May 1987.

[18] Graham Hutton. *A simple guide to the Ruby simulator.* Glasgow University, August 1991.

[19] Geraint Jones. *Designing circuits by calculation.* Oxford University, April 1990. (PRG–TR–10–90)

[20] G. Jones and M. Sheeran. *Designing arithmetic circuits by calculation.* Glasgow University, 1991. [In preparation]

[21] Proc. EURICS Workshop on Calculational Theories of Program Structure, Ameland, The Netherlands, September 1991.