Abstract Interpretations for Free

Kevin Backhouse* Roland Backhouse†

January 15, 2002

Abstract

A construction is given for extending Galois connections between base types (eg booleans and integers) to Galois connections between datatypes of arbitrary complexity. "Theorems-for-free" is used to show that the construction ensures safe abstract interpretation of parametrically polymorphic functions.

1 Introduction

Abstract interpretation is a technique for approximating the execution behaviour of a computer program. It is used, for example, for strictness analysis of functional programs; the results of such an analysis are approximate in the sense that non-strict programs will always be correctly identified as such but strict programs may be incorrectly classified. In this sense the approximation is "safe". The basis of abstract interpretation is the notion of a Galois connection between ordered sets [CC77, CC79].

"Theorems-for-free" [Rey83, Wad89] is a remarkable feature of parametrically polymorphic functions, namely that knowing just the type of a function it is possible to derive algebraic properties of the function. For example, it is possible to show that any function that has the same type as function composition must be associative.

In this paper we show that safety properties of abstract interpretations are "free theorems". The basis of this result is a construction for extending a collection of Galois connections on base types (eg booleans and integers) to Galois connections between datatypes of arbitrary complexity. This construction is then combined with "theorems-for-free" to establish the safety property.

*Computing Laboratory, University of Oxford, Oxford OX3 9QD, England, email: kevinb@comlab.ox.ac.uk, fax: +44(0) 1865 273839

†School of Computer Science and Information Technology, University of Nottingham, Nottingham NG8 1BB, England, email: rcb@cs.nott.ac.uk, fax: +44(0) 115 951 4254

There are several practical applications of this result. The theorem can be used to construct abstract interpretations. The theorem can also be used to focus the testing of programs (as in model checking) on those properties that are not for free. Most importantly, the theorem suggests a design principle for the development of programs involving a separation into generic and non-generic components. This latter aspect is illustrated briefly in section 6.

Connections between abstract interpretations and "theorems-for-free" —more specifically, so-called "logical relations"— have been observed by Abramsky [Abr90]; he proves a theorem (the "uniformization theorem") which is similar to our extension of base Galois connections to Galois connections between datatypes of arbitrary complexity. Also, a generic safety property of abstract interpretations has been proved by Burn, Hankin and Abramsky [BHA86]. However, so far as we can determine, the proof of safety properties by exploiting "theorems-for-free" is novel.

We have chosen in this paper to relegate proofs to an appendix. This has been done for two reasons. First, we want to emphasise the applicability of the results. Second, the beautifully concise relation-algebraic proofs are relatively unfamiliar in the literature and, for this reason, require some preparatory explanation before they can be presented. We have also chosen to give brief introductions to both abstract interpretations and to theorems-for-free in the hope that the paper will also be accessible to readers familiar with one but not the other (and possibly readers familiar with neither). The examples we develop in these introductions are expanded later.

2 Running Example

We start by introducing a simple, running example. Consider the fold function which, given a starting value m of some type a, a binary operator $\oslash$ of type $a \leftarrow a \times a$ and a list $[m_0, \ldots, m_{n-1}]$ of a's, evaluates $((m \oslash m_0) \oslash \ldots) \oslash m_{n-1}$. So $\mathit{fold}.0.(+).ms$ sums a list ms of integer values, $\mathit{fold}.1.(\times).ms$ evaluates the product of the list of values, $\mathit{fold}.\mathit{true}.(\equiv).bs$ evaluates the associative (as opposed to conjunctional) equivalence of the list of booleans bs, and $\mathit{fold}.\mathit{false}.(\vee).bs$ evaluates whether one of the values in the list bs is true.

Now suppose we want to determine whether some fold on a list of integer values evaluates to a number that is even without actually evaluating the fold computation proper. For example, we may want to determine whether

$$18975 \times 3 \times 2 \times 21$$

is even without performing the integer division. "Theorems-for-free" predicts that this can be done by evaluating

$$even.18975 \vee even.3 \vee even.2 \vee even.21 \quad .$$

Formally, a "theorem for free" is

$$even.(fold.1.(\times).ms) \;=\; fold.false.(\vee).(List.even.ms)$$

where $List.even.ms$ maps the function $even$ over the list of numbers ms. ($List$ is normally denoted by map in functional languages. As we want to generalise $List$ to an arbitrary datatype F it is useful to adopt the categorical convention of using the same symbol to denote the type constructor and its associated map operation.) Similarly, "theorems-for-free" also predicts that

$$even.(18975{+}3{+}2{+}21) \;=\; (even.18975 \equiv even.3 \equiv even.2 \equiv even.21) \quad .$$

Formally,

$$even.(fold.0.(+).ms) \;=\; fold.true.(\equiv).(List.even.ms) \quad .$$

That is, a sum of numbers is even if the (associative) equivalence of their individual parities is even.

That these properties of $fold$ are applications of theorems-for-free is, of course, well-known. (For $fold$ the general theorem is often called the *fusion* theorem.)

Suppose now that we want to determine whether a product of numbers is divisible by k for some given positive number k. For brevity, define the function dk mapping integers to booleans by

$$dk.m \;=\; m/k$$

where m/k is read as "m is divisible by k". Then we will show that theorems-for-free predicts that

$$dk.(fold.1.(\times).ms) \;\Leftarrow\; fold.(k{=}1).(\otimes).(List.dk.ms)$$

where

$$b \otimes c \;=\; (k{=}1) \vee b \vee c \quad .$$

That is, the product of a list of numbers is divisible by k if $k{=}1$ or one of the numbers is divisible by k.

Note that the theorem-for-free here is an "if" ($\Leftarrow$) rather than an equality. We say that $fold.(k{=}1).(\otimes).(List.dk.ms)$ is an *abstract interpretation* of $dk.(fold.1.(\times).ms)$ and its result is *safe* in the sense that if the abstract interpretation evaluates to $true$ then it is the case that the given product is divisible by k; however, if the abstract interpretation evaluates to $false$ then it is not known whether or not the given product

is divisible by k. For example, taking k to be 4 (the smallest non-prime positive number) the above abstract interpretation of 2×2 evaluates to $false$ whereas 2×2 is of course divisible by 4.

Our running example is *deliberately* simple. The reader will have no difficulty in seeing how to evaluate divisibility by k, for arbitrary k, but that is not the point of the example. Abstract interpretations are typically concerned with computations that are so complex, or perhaps even impossible, that a safe result is all that one can expect to achieve. Moreover, we are concerned here with a very general result; the example we have chosen —the combination of the function $fold$ with the test for divisibility by k— combines the two basic ingredients of this general result, namely theorems-for-free and Galois connections. Let us now introduce the basics needed to understand these concepts.

3 Preliminaries

Galois Connections and Pair Algebras Galois connections are most often defined in terms of a pair of functions. A better starting point for a discussion of Galois connections is, arguably, relations, as proposed by Hartmanis and Stearns [HS64, HS66] in what is most probably the very first application of Galois connections in the computing science literature.

A binary relation R on the posets $(A, \sqsubseteq)$ and $(B, \preceq)$ is called a *pair algebra* if there are functions $f \in A \leftarrow B$ and $f^\sharp \in B \leftarrow A$ such that, for all x in B and all y in A,

$$(x, y) \in R \quad \equiv \quad f.x \sqsubseteq y$$

and

$$(x, y) \in R \quad \equiv \quad x \preceq f^\sharp.y \quad .$$

We say that the pair of functions $(f, f^\sharp)$ is *a Galois connection between the posets* $(A, \sqsubseteq)$ *and* $(B, \preceq)$ if $f \in A \leftarrow B$ and $f^\sharp \in B \leftarrow A$ and, for all x in B and all y in A,

$$f.x \sqsubseteq y \quad \equiv \quad x \preceq f^\sharp.y \quad .$$

Clearly a Galois connection defines a pair algebra and a pair algebra defines a Galois connection. A major element of the proofs in this paper is that we instantiate "theorems for free" with the pair algebra defined by a Galois connection.

The Galois connection involved in our running example is the pair (dk, kd) between $(Bool, \Leftarrow)$ and $(PosInt, /)$ where $dk \in Bool \leftarrow PosInt$, $kd \in PosInt \leftarrow Bool$ and $/$ is the is-divisible-by ordering on positive integers. Specifically, kd is defined by, for all booleans b,

$$kd.b \;=\; \textbf{if } b \textbf{ then } k \textbf{ else } 1$$

and satisfies, for all positive integers m and all booleans b,

$$dk.m \Leftarrow b \quad \equiv \quad m \,/\, kd.b \quad .$$

(Read "(m is divisible by k if b) is (m is divisible by $kd.b$)".) The pair algebra corresponding to this Galois connection relates all positive integers to the boolean $false$ and the positive integers divisible by k to the boolean $true$.

Theorems for Free Based on Reynolds' abstraction theorem [Rey83], Wadler [Wad89] showed how to derive a theorem about a polymorphic function from its type. The key to such "theorems for free" is, in Wadler's words, "that types may be read as relations". Briefly, the type A is read as the identity relation id_A on A and the function space constructor "$\leftarrow$" is read as a mapping from a pair of relations R and S of types $A \leftarrow B$ and $C \leftarrow D$, respectively, to a binary relation $R \leftarrow S$ on functions f and g of type $A \leftarrow C$ and $B \leftarrow D$, respectively. Formally, $R \leftarrow S$ is defined by

$$(1) \qquad (f, g) \in R \leftarrow S \quad \equiv \quad \langle \forall u,v :: (f.u\,,\, g.v) \in R \;\Leftarrow\; (u, v) \in S \rangle \quad .$$

In words, f and g construct R-related values from S-related values.

As an example, suppose $(A\,, \sqsubseteq)$ and $(B\,, \preceq)$ are partially ordered sets. Then we can instantiate R to $\sqsubseteq$ and S to $\preceq$ getting a relation $\sqsubseteq \leftarrow \preceq$ between functions f and g of type $A \leftarrow B$. In particular, switching to the usual infix notation for membership of an ordering relation,

$$(f, f) \;\in\; \sqsubseteq \leftarrow \preceq \quad \equiv \quad \langle \forall u,v :: f.u \sqsubseteq f.v \Leftarrow u \preceq v \rangle \quad .$$

So $(f, f) \in \sqsubseteq \leftarrow \preceq$ is the statement that f is a monotonic function. In Wadler's words, f maps $\preceq$-related values to $\sqsubseteq$-related values.

It is useful to abbreviate a property of the form $(x, x) \in R$ to $x \in R$. In this way, the statement $f \in \sqsubseteq \leftarrow \preceq$ means that f is a monotonic function, the ordering relations on the range and domain of f being, respectively, $\sqsubseteq$ and $\preceq$.

Particularly relevant to this paper is that, if f and g are both *monotonic* functions of type $A \leftarrow B$ where $(A\,, \sqsubseteq)$ and $(B\,, \preceq)$ are posets,

$$(f, g) \;\in\; \sqsubseteq \leftarrow \preceq \quad \equiv \quad \langle \forall u :: f.u \sqsubseteq g.u \rangle \quad .$$

(The easy proof is left to the reader.) We thus recognize $\sqsubseteq \leftarrow \preceq$ as the usual pointwise ordering on monotonic functions of type $(A\,, \sqsubseteq) \leftarrow (B\,, \preceq)$.

"Theorems for free" is the property that, if f is a parametrically polymorphic function of type t, where t is a type expression parameterised by type variables $a, \ldots, c$, and, for each type variable a, R_a is a relation then $f \in t[a, \ldots, c := R_a\,, \ldots\,, R_c]$. An example of a "free theorem" is obtained by considering the composition operator on functions, which we denote by a raised infix dot as in "$f \bullet g$". Composition has parametric type

$$(a \leftarrow b) \leftarrow (a \leftarrow c) \leftarrow (c \leftarrow b) \quad .$$

The free theorem is that, for all relations R, S and T,

$$\bullet \in (R \leftarrow S) \leftarrow (R \leftarrow T) \leftarrow (T \leftarrow S) \quad .$$

In particular, substituting ordering relations $\sqsubseteq$, $\preceq$ and $\leq$ for R, S and T, respectively,

$$\bullet \in (\sqsubseteq \leftarrow \preceq) \leftarrow (\sqsubseteq \leftarrow \leq) \leftarrow (\leq \leftarrow \preceq) \quad .$$

Spelling out the definition of the arrow operator, this has the corollary that

$$f \bullet g \in \sqsubseteq \leftarrow \preceq \quad \Leftarrow \quad f \in \sqsubseteq \leftarrow \leq \quad \wedge \quad g \in \leq \leftarrow \preceq \quad .$$

In words, $f \bullet g$ is a monotonic function if both f and g are monotonic. Note that this property is derived solely from the type of function composition. So "theorems for free" says that any function that has the same type as function composition will preserve the monotonicity property.

Note that, if we were to be completely precise, we should distinguish between the different instances of function composition. Also, if θ is a parametrically polymorphic function of type t then the free theorem is actually that the pair $(\theta(A), \theta(B))$ is an element of relation $t[a, \ldots, c := R_a, \ldots, R_c]$ where A and B indicate how the particular instances of θ are determined (depending on the types of the relations R). We omit these details for the moment but include them in the formal statements of our theorems.

An instance of $R \leftarrow S$ that is used extensively in this paper is when R is a function and S is the converse of a function[1]. For functions f and g, $f \leftarrow g^{\cup}$ is the function defined by

$$(f \leftarrow g^{\cup}).h \quad = \quad f \bullet h \bullet g \quad .$$

(The easy proof is left to the reader.)

Relators Another element of "theorems for free" is that type constructors, like disjoint sum $+$, cartesian product $\times$ and $List$ have to be extended to map relations to relations. This led Roland Backhouse to propose the notion of a *relator*[2] as the basis for a relational theory of datatypes [BBH+92, BVW92]. Briefly, a relator is a monotonic functor that commutes with converse. Formally, a relator, F, is a pair of mappings from a source allegory to a target allegory. The first element of the pair is from objects (types) of the

[1] Functions are regarded as special cases of binary relations. For function f, $(x, y) \in f$ is the same as $x = f.y$. The converse $g^{\cup}$ of function g is thus such that $(x, y) \in g^{\cup}$ is the same as $y = g.x$.

[2] Not be confused with Abramsky and Jensen's [AJ90] and similar proposals made at about the same time.

source allegory to objects of the target allegory and the second element is from arrows (relations) of the source allegory to arrows of the target allegory. The mappings are required to have the properties that they form a functor from the underlying source category to the underlying target category and, for all arrows R and S,

$$F.R \subseteq F.S \;\Leftarrow\; R \subseteq S \quad ,$$

and

$$(F.R)^{\cup} \;=\; F.(R^{\cup}) \quad .$$

An example of a unary relator is $List$. Suppose R relates values of type A to values of type B. Then $List.R$ relates values of type $List.A$ to values of type $List.B$. Specifically, the list as is related to the list bs by $List.R$ if as and bs have the same length and corresponding elements are related by R. An example of a binary relator is cartesian product. Suppose R relates values of type A to values of type B and S relates values of type C to values of type D. Then $R \times S$ relates values of type $A \times C$ to values of type $B \times D$. Specifically, the pair (a, c) is related to the pair (b, d) by $R \times S$ if a is related by R to b and c is related by S to d. An example of a zero-ary relator is $\mathbb{N}$; this relates two natural numbers exactly when they are equal. (Any type defines a zero-ary relator in this way.)

By design, it is easy to show that relators preserve functions. That is, if f is a function then $F.f$ is also a function.

Note that the formal definition of a relator encompasses the possibility of relators with different source and target allegories. We will be somewhat informal in our treatment of relators. Typically, we assume that the source allegory is the n-fold product of the target allegory with itself, for some unspecified number n $(n \geq 0)$ and will write $F.(u \ldots v)$ for the application of the relator to arguments u, ..., v. Examples used later are the unary relator $List$, the binary relator $\times$, and a zero-ary relator $Production$. (Every object in an allegory defines a zero-ary relator. On arrows the relator maps each arrow to the identity arrow on the object.)

In this paper, we will also require that relators preserve partial orderings (so that if R is reflexive, antisymmetric and transitive then so too is $F.R$). For this we need the additional property that relators distribute through binary intersections. That is, we assume that, for all relations R and S, $F.(R \cap S) = F.R \cap F.S$. This assumption is satisfied by all "regular" datatypes. Indeed, it is easily verified for disjoint sum and cartesian product and then a straightforward proof shows that the property is preserved by the usual fixpoint construction of datatypes like $List$. So the assumption is a reasonable one in the context of modern programming languages.

For more background on relators see [BBH+92, BVW92, BdM96]. For a formal treatment of relators in their full generality see [Hoo97].

Running Example Returning to our running example, the function $fold$ has type

$$a \leftarrow a \leftarrow (a \leftarrow a \times a) \leftarrow List.a$$

for all instances of the type variable a. (Actually it is more general than this but this is all we need for the purposes of the example.) Theorems-for-free predicts that, for arbitrary relation P

$$fold \in P \leftarrow P \leftarrow (P \leftarrow P \times P) \leftarrow List.P \quad .$$

In particular, we can take for P the pair algebra of the Galois connection (dk, kd) mentioned above.

The theorem we present predicts first that the function

$$dk \leftarrow kd^{\cup} \leftarrow (kd^{\cup} \leftarrow dk \times dk) \leftarrow List.kd^{\cup}$$

is the lower adjoint in a Galois connection between the poset of monotonic functions of type

$$Bool \leftarrow Bool \leftarrow (Bool \leftarrow Bool \times Bool) \leftarrow List.Bool$$

(ordered pointwise by $\Leftarrow$) and monotonic functions of type

$$PosInt \leftarrow PosInt \leftarrow (PosInt \leftarrow PosInt \times PosInt) \leftarrow List.PosInt$$

(ordered pointwise by divisibility). The upper adjoint is the function

$$kd \leftarrow dk^{\cup} \leftarrow (dk^{\cup} \leftarrow kd \times kd) \leftarrow List.dk^{\cup} \quad .$$

Our second theorem says that the free theorem for $fold$ instantiated with the pair algebra for the (dk, kd) Galois connection has the corollary that, for all binary operators $\otimes \in PosInt \leftarrow PosInt \times PosInt$ that are monotonic with respect to divisibility, all $m \in PosInt$ and all $ms \in List.PosInt$,

$$dk.ConcreteValue \Leftarrow AbstractValue$$

where

$$ConcreteValue = fold.m.(\otimes).ms$$

and

$$AbstractValue = fold.(dk.m).((dk \leftarrow kd^{\cup} \times kd^{\cup}).(\otimes)).(List.dk.ms) \quad .$$

Substituting integer multiplication[3] for $\otimes$ we calculate that, for all boolean values b and c,

[3] denoted here by the symbol "$*$" in order to distinguish it from "$\times$", meaning cartesian product

$$
\begin{array}{ll}
 & (dk \leftarrow kd^{\cup} \times kd^{\cup}).(*).(b,c) \\
= & \quad \{ \quad \text{definitions} \quad \} \\
 & dk.(kd.b * kd.c) \\
= & \quad \{ \quad kd.true = k\,,\ kd.false = 1\,, \\
 & \qquad\qquad dk.(1*1) = (k=1)\,,\ dk.(1*k) = dk.(k*1) = dk.(k*k) = true \quad \} \\
 & (k=1) \vee b \vee c \quad .
\end{array}
$$

In this way we have calculated a safe abstract interpretation determining whether a sequence of integer multiplications results in a value divisible by k.

Let us now present the theorems formally.

4 Types and Assignments to Variables

We consider the following (extended BNF) grammar of type expressions

$$TypeExp \quad ::= \quad (Relator\ TypeExp^{*}) \mid (TypeExp \leftarrow TypeExp) \mid Variable$$

where $Variable$ and $Relator$ are (disjoint) finite sets. Variables are place-holders to be filled in various ways, as explained in detail later. Each element of $Relator$ denotes a relator of a certain arity. Examples include $\mathbb{1}$, which denotes the relator of arity zero corresponding to the unit type, and $List$, which denotes the unary relator corresponding to the list type constructor. Binary relators include cartesian product and disjoint sum.

According to the syntax above, the application of a binary relator, say $+$, to types t and u would be denoted by the prefix notation $+tu$. We will, however, deviate from the formal syntax in our examples and write $t+u$ instead. Also, the formal syntax stipulates that subtypes are parenthesised (in order to avoid specifying precedence and associativity rules); again we will ignore this requirement when presenting examples. Instead we assume that $\leftarrow$ has the lowest precedence and associates to the left.

An example of a type expression is

$$a \times List.b \leftarrow b \times List.a \leftarrow Production \quad .$$

This uses the binary product relator, $\times$, the unary relator $List$ and the zero-ary relator $Production$. It describes the type of a semantic rule in an attribute grammar, with a and b being place-holders for the types of the synthesised and inherited attributes, respectively. (A semantic rule specifies, for each production in the attribute grammar, a function from an inherited attribute and a list of synthesised attributes to a synthesised attribute and a list of inherited attributes.)

A *variable assignment* is a function with domain $Variable$, the set of type variables. We consider two kinds of variable assignments, one with range posets and the other with range (binary) relations. A variable assignment will be denoted by an assignment statement as in, for example,

$$a,b := integer,boolean \quad .$$

Given a variable assignment V, its application to type variable a is denoted by V_a.

In the case of variable assignments whose ranges are sets of relations we need to define the converse and composition operators. The definitions are:

$$(2) \qquad (V^{\cup})_a \;=\; (V_a)^{\cup}$$

and

$$(3) \qquad (V \bullet W)_a \;=\; V_a \bullet W_a \quad .$$

We extend variable assignments to type expressions in two ways. The simpler of the two extends a variable assignment V inductively to all type expressions t: for all type expressions u, v

$$(4) \qquad V_{u \leftarrow v} \;=\; V_u \leftarrow V_v$$

and, for all type expressions u, ..., v,

$$(5) \qquad V_{F.(u \ldots v)} \;=\; F.(V_u \ldots V_v) \quad .$$

Note that these definitions make sense whenever the range of V is an assignment of posets or relations to the type variables. In the case that V is a poset assignment we define the $\leftarrow$ operator to map posets A and B into the *monotonic* functions mapping values of poset B into values of poset A.

The more complicated extension uses two assignments, one for the positive and the other for the negative occurrences of type variables. Specifically, suppose V and W are two variable assignments of the same kind. Then the variable assignment $[V,W]$ is defined inductively as follows:

$$(6) \qquad [V,W]_a \;=\; V_a \quad ,$$

$$(7) \qquad [V,W]_{u \leftarrow v} \;=\; [V,W]_u \leftarrow [W,V]_v \quad ,$$

$$(8) \qquad [V,W]_{F.(u \ldots v)} \;=\; F.([V,W]_u \ldots [V,W]_v) \quad .$$

For example,

$$[(a := dk),(a := kd^{\cup})]_{a \leftarrow a \leftarrow (a \leftarrow a \times a) \leftarrow List.a}$$
$$= dk \leftarrow kd^{\cup} \leftarrow (kd^{\cup} \leftarrow dk \times dk) \leftarrow List.kd^{\cup} \quad .$$

From the above definitions, the following properties are easily proved for all type expressions t by induction on the structure of type expressions.

(9) $\quad V_t = [V,V]_t \quad ,$

(10) $\quad (V_t)^{\cup} = (V^{\cup})_t \quad ,$

(11) $\quad ([V,W]_t)^{\cup} = \left[V^{\cup}, W^{\cup}\right]_t \quad ,$

Because of (10) we write $V_t^{\cup}$, omitting the parentheses. We also write $[V,W]_t^{\cup}$; here the justification is that we define $[V,W]^{\cup}$ to be $[V^{\cup}, W^{\cup}]$.

Note that it is not the case that $(V \bullet W)_t = V_t \bullet W_t$ for all type expressions t. Property (3) is only valid for type variables a.

5 Main Theorem

Let us suppose that an assignment of Galois connections (abs_a, con_a) between the posets $(A_a, \sqsubseteq_a)$ and $(B_a, \preceq_a)$ is given for each type variable a. In other words, we are given six functions A, B, $\sqsubseteq$, $\preceq$, abs and con on the type variables. Extending them to type expressions as detailed in section 4, we have the following theorems.

Theorem 12 (Galois Connections For Free) For all type expressions t, $(A_t, \sqsubseteq_t)$ is a partially ordered set, as is $(B_t, \preceq_t)$; moreover, the pair $([abs, con^{\cup}]_t \ , \ [con, abs^{\cup}]_t)$ is a Galois connection between these two posets.

□

Returning to our running example, an instance of theorem 12 is that the pair of functions

$$dk \leftarrow kd^{\cup} \leftarrow (kd^{\cup} \leftarrow dk \times dk) \leftarrow List.kd^{\cup}$$

and

$$kd \leftarrow dk^{\cup} \leftarrow (dk^{\cup} \leftarrow kd \times kd) \leftarrow List.dk^{\cup}$$

is a Galois connection between the posets

$$Bool \leftarrow Bool \leftarrow (Bool \leftarrow Bool \times Bool) \leftarrow List.Bool$$

ordered by

$$(\Leftarrow) \leftarrow (\Leftarrow) \leftarrow ((\Leftarrow) \leftarrow (\Leftarrow) \times (\Leftarrow)) \leftarrow \mathit{List}.(\Leftarrow)$$

and

$$\mathsf{PosInt} \leftarrow \mathsf{PosInt} \leftarrow (\mathsf{PosInt} \leftarrow \mathsf{PosInt} \times \mathsf{PosInt}) \leftarrow \mathit{List}.\mathsf{PosInt}$$

ordered by

$$(/) \leftarrow (/) \leftarrow ((/) \leftarrow (/) \times (/)) \leftarrow \mathit{List}.(/) \quad .$$

Theorem 13 Define the variable assignment P by

$$P_a \;=\; abs_a^{\cup} \bullet \sqsubseteq_a \quad .$$

Let P_t be its extension to arbitrary type expression t, as detailed above. Then, for all type expressions t,

$$P_t \;=\; [abs, con^{\cup}]_t^{\cup} \bullet \sqsubseteq_t \quad .$$

That is, x and y are related by the relation P_t obtained by extending the pair algebras P_a to arbitrary type expressions exactly when x and y are related by the pair algebra defined by the constructed Galois connection.

□

Corollary 14 (Safety For Free) If expression e has type t and is parametrically polymorphic in all the type variables in t then

$$[abs, con^{\cup}]_t.e \;\sqsubseteq_t\; e \quad .$$

(The two occurrences of e are, of course, different instances of the given expression. See the discussion of the free theorem above.)

Proof The assumption that e has type t and is parametrically polymorphic in all the type variables in t means that

$$e \in P_t \quad .$$

The corollary is thus immediate from theorem 13.
□

Applying theorem 12 to our running example, we know that, for all a, given a pair algebra P_a,

$$\mathit{fold} \;\in\; P_a \leftarrow a \leftarrow (a \leftarrow a \times a) \leftarrow \mathit{List}.a \quad .$$

This is true in particular for P_a defined to be $dk^{\cup} \bullet (\Leftarrow)$. Corollary 14 gives the property mentioned earlier, namely

$$dk.(fold.m.(\times).ms) \;\Leftarrow\; fold.(dk.m).(\otimes).(List.dk.ms)$$

where

$$b \otimes c \;=\; (k = 1) \vee b \vee c \quad .$$

Specifically, the corollary states that

$$(dk \leftarrow kd^{\cup} \leftarrow (kd^{\cup} \leftarrow dk \times dk) \leftarrow List.kd^{\cup}).fold \;\Leftarrow\; fold$$

where $\Leftarrow$ is the pointwise extension of the $\Leftarrow$ ordering on boolean-valued functions. Using the algebraic properties of the arrow operator this point-free formulation of the property of $fold$ instantiates to the above pointwise property in the case that the binary operator supplied as parameter to $fold$ is multiplication (which is, of course, monotonic with respect to divisibility).

6 CSP Traces

This section illustrates how our theorems might be applied to programming language semantics. The (much simplified) example is based on Hoare's CSP language [Hoa85] restricted to just choice and concurrency. Here are two simple processes, written in Hoare's notation:

$$\begin{array}{lcl} Customer & = & coin \rightarrow choc \rightarrow Customer \\ Vendor & = & coin \rightarrow (choc \rightarrow Vendor \mid toffee \rightarrow Vendor) \end{array}$$

Identifiers beginning with a capital letter denote *processes*; the other identifiers denote *events*. If P is a process and x an event, then $x \rightarrow P$ is a process that first engages in the event x and then behaves like P. The process $(x \rightarrow P \mid y \rightarrow Q)$ offers a choice. Either it engages in x and then continues as P or it engages in y and then continues as Q. Hoare generalises these constructions to the general *choice* operator $(x : B \rightarrow P(x))$. Here B is a finite set of events. The process first engages in one of the events in B and then continues as $P(x)$.

A second method of constructing processes is *concurrency*. The process $P \| Q$ is the synchronisation of the processes P and Q. That is, an event can only happen if both P and Q can simultaneously engage in that event. For example, we might want to run the $Customer$ and $Vendor$ processes concurrently. This would lead to a process that is equivalent to the $Customer$ process, because the $Vendor$ process is prepared to vend toffee as well as chocolate, but the customer only ever requests chocolate.

6.1 The interface to the library

We first define an interface for a simple CSP library. The interface contains two methods, the general choice operator and the concurrency operator. These are parameterised by the type of processes, τ, and the type of events, α, in order that they can be given different interpretations at a later stage.

$$\mathbf{type}\ \mathrm{CSP} = \left\langle \forall \tau,\ \alpha :: \begin{array}{lcl} \mathrm{choice} & : & \tau \leftarrow \mathrm{List}.(\alpha \times \tau) \\ (\ \|\ ) & : & \tau \leftarrow \tau \times \tau \end{array} \right\rangle$$

The definition of $\rightarrow$ is:

$$\begin{array}{lcl} (\ \rightarrow\ ) & :: & \langle \forall \tau,\ \alpha :: \ \tau \leftarrow \tau \leftarrow \alpha \leftarrow \mathrm{CSP}.\tau.\alpha \rangle \\ x \rightarrow_C P & = & \mathrm{choice}_C.[(x, P)] \end{array}$$

Let us now define the $\mathrm{Customer}$ and Vendor processes with our library. First we need to define the type of events:

$$\mathbf{type}\ A = \mathrm{coin} \mid \mathrm{choc} \mid \mathrm{toffee}$$

Then the definitions of the processes are:

$$\begin{array}{lcl} \mathrm{Customer} & :: & \langle \forall \tau :: \ \tau \leftarrow \mathrm{CSP}.\tau.A \rangle \\ \mathrm{Customer}_C & = & \mathrm{coin} \rightarrow_C \mathrm{choc} \rightarrow_C \mathrm{Customer}_C \end{array}$$

$$\begin{array}{lcl} \mathrm{Vendor} & :: & \langle \forall \tau :: \ \tau \leftarrow \mathrm{CSP}.\tau.A \rangle \\ \mathrm{Vendor}_C & = & \mathrm{coin} \rightarrow_C (\mathrm{choice}_C\ [(\mathrm{choc}, \mathrm{Vendor}_C), (\mathrm{toffee}, \mathrm{Vendor}_C)]) \end{array}$$

Note that these processes are not polymorphic in the type of events, because they explicitly refer to the events coin, choc and toffee. However they are still polymorphic in τ, because they are still independent of the implementation of the library.

6.2 Traces

Hoare defines the semantics of CSP using *traces*. A trace t of the process P is a finite sequence of events that P might engage in. We will use lists to represent traces. Traces are partially ordered:

$$(15) \qquad s \leq t \ \equiv\ \langle \exists u \ ::\ \ s +\!\!+ u = t \rangle \quad .$$

The semantics of a process P is the set of all traces of P. This leads us to define the type of processes as:

$$\mathbf{type}\ \mathrm{Proc} = \langle \forall \alpha \ ::\ \ \mathcal{P}.(\mathrm{List}.\alpha) \rangle \quad .$$

That is, if events have type α, then processes are sets of traces of α. If the set A of all events is finite, then $Proc.A$ is a complete lattice under the set inclusion ($\subseteq$) ordering. The bottom element of the lattice is the empty set $\emptyset$ and the top element is $all.A$, where all is defined as follows:

$$\begin{array}{lcl} all & :: & \langle \forall \alpha \ :: \ Proc.\alpha \leftarrow \mathcal{P}.\alpha \rangle \\ all.X & = & \{s \mid elems.s \subseteq X\} \end{array}$$

(In the notation of regular expressions $all.X$ is X^*.) The function elems calculates the set of events that appear in a trace. Its definition is:

$$\begin{array}{lcl} elems & :: & \langle \forall \alpha \ :: \ \mathcal{P}.\alpha \leftarrow List.\alpha \rangle \\ elems.[] & = & \emptyset \\ elems.(x{:}xs) & = & \{x\} \cup elems.xs \end{array}$$

Using this definition of a process, we can implement the CSP interface:

$$Tr \ :: \ \langle \forall \alpha \ :: \ CSP.(Proc.\alpha).\alpha \rangle$$

$$Tr \ = \ \left[\begin{array}{lcl} choice.[] & = & \{[]\} \\ choice.((x,P){:}cs) & = & \{[]\} \cup \{x{:}s \mid s \in P\} \cup choice.cs \\ & & \\ P \| Q & = & P \cap Q \end{array} \right.$$

We can use Tr to evaluate some of the processes that we defined earlier. For example, if we evaluate the $Customer$ process, we find that:

$$Customer_{Tr} = \{[]\} \cup \{coin{:}choc{:}s \mid s \in Customer_{Tr}\}$$

In other words, $Customer_{Tr}$ is the set $\{[], [coin, choc], \ldots\}$ as expected.

6.3 Analysing the alphabet

The $Customer$ and $Vendor$ processes are very simple, so it is easy for us to see which events they might engage in. $Customer$ can only engage in $coin$ and $choc$, whereas $Vendor$ is also capable of engaging in $toffee$. However, in a larger system it may be useful to have a tool that can do this analysis automatically.

If we are given the set of traces of a process, then we can use the function $events$ to find the set of events that the process might engage in:

$$\begin{array}{lcl} events & :: & \langle \forall \alpha \ :: \ \mathcal{P}.\alpha \leftarrow Proc.\alpha \rangle \\ events.P & = & \langle \bigcup s : \ s \in P \ : \ elems.s \rangle \end{array}$$

Notice that $events$ forms a Galois connection with all:

$$events.P \subseteq X \quad \equiv \quad P \subseteq all.X \quad .$$

This means that we can derive a new library Ev that calculates the set of events that a process engages in. From the types of choice and concurrency our theorem predicts that the libraries Tr and Ev should be related as follows:

$$\begin{array}{lcl} choice_{Ev} & = & (events \leftarrow List.(id \times all^{\cup})).choice_{Tr} \\ (\ \|_{Ev}\ ) & = & (events \leftarrow all^{\cup} \times all^{\cup}).(\ \|_{Tr}\ ) \end{array}$$

Using the definition of the $\leftarrow$ operator this reduces to:

$$\begin{array}{lcl} choice_{Ev} & = & events \bullet choice_{Tr} \bullet List.(id \times all) \\ (\ \|_{Ev}\ ) & = & events \bullet (\ \|_{Tr}\ ) \bullet (all \times all) \end{array}$$

Hence, the definition of Ev is:

$$Ev :: \langle \forall \alpha :: CSP.(\mathcal{P}.\alpha).\alpha \rangle$$

$$Ev = \left\lceil \begin{array}{lcl} choice.[] & = & \emptyset \\ choice.((x,P):cs) & = & \{x\} \cup P \cup choice.cs \\ & & \\ P \| Q & = & P \cap Q \end{array} \right.$$

Using this library, we evaluate the $Customer$ process:

$$Customer_{Ev} = \{coin\} \cup \{choc\} \cup Customer_{Ev}$$

In other words, $Customer_{Ev} = \{coin, choc\}$, as expected. If we apply the library to $Vendor$, then we find that $Vendor_{Ev} = \{coin, choc, toffee\}$, which is also as expected.

References

[Abr90] Samson Abramsky. Abstract interpretation, logical relations, and kan extensions. *J. Logic and Computation*, 1(1):5–41, 1990.

[AJ90] Samson Abramsky and Thomas P. Jensen. A relational approach to strictness analysis for higher-order polymorphic functions. In *???*, 1990.

[Bac90] R.C. Backhouse. On a relation on functions. In W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty is our Business*. Springer-Verlag, 1990.

[BBH+92] R.C. Backhouse, P. de Bruin, P. Hoogendijk, G. Malcolm, T.S. Voermans, and J. van der Woude. Polynomial relators. In M. Nivat, C.S. Rattray, T. Rus, and G. Scollo, editors, *Proceedings of the 2nd Conference on Algebraic Methodology and Software Technology, AMAST'91*, pages 303–326. Springer-Verlag, Workshops in Computing, 1992.

[BdM96] Richard S. Bird and Oege de Moor. *Algebra of Programming*. Prentice-Hall International, 1996.

[BHA86] G.L. Burn, C.L. Hankin, and S. Abramsky. The theory of strictness analysis for higher order functions. *Science of Computer Programming*, 7:249–278, 1986.

[BVW92] R.C. Backhouse, T.S. Voermans, and J. van der Woude. A relational theory of datatypes. Available via World-Wide Web at `http://www.cs.nott.ac.uk/~rcb/MPC/papers` Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`, December 1992.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unifed lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, January 1977.

[CC79] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, January 1979.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hoo97] Paul Hoogendijk. *A Generic Theory of Datatypes*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1997.

[HS64] J. Hartmanis and R.E. Stearns. Pair algebras and their application to automata theory. *Information and Control*, 7(4):485–507, 1964.

[HS66] J. Hartmanis and R.E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, 1966.

[NNH98] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1998.

[Rey83] J.C. Reynolds. Types, abstraction and parametric polymorphism. In R.E. Mason, editor, *IFIP '83*, pages 513–523. Elsevier Science Publishers, 1983.

[Wad89] P. Wadler. Theorems for free! In *4'th Symposium on Functional Programming Languages and Computer Architecture, ACM, London*, September 1989.

7 Appendix

The proofs of theorems 12 and 13 use point-free relation algebra. This has the advantage of reducing the size of the formulae we have to manipulate, making the proofs easier to construct and to verify. On the other hand, point-free relation algebra is not widely known so we need to begin by introducing some basic rules.

Types For the purposes of this paper we assume that types are partially ordered sets. Supposing $(A, \sqsubseteq)$ and $(B, \preceq)$ are posets, $(A, \sqsubseteq) \leftarrow (B, \preceq)$ denotes the set of *monotonic* functions with range A and domain B ordered pointwise.

Relation Algebra A *binary relation of type* $A \sim B$ is a subset of $A \times B$. We use the usual infix operators $\supseteq$ and $\subseteq$ to denote set inclusion and containment. Given binary relations $R \in A \sim B$ and $S \in B \sim C$, $R \bullet S$ denotes their composition, the relation of type $A \sim C$ defined by

$$(16) \qquad (x, z) \in R \bullet S \quad \equiv \quad \langle \exists y :: (x, y) \in R \wedge (y, z) \in S \rangle \quad .$$

We use $^{\cup}$ as a postfix operator to denote the converse operation on relations. So, if $R \in A \sim B$ then $R^{\cup}$ is the relation of type $B \sim A$ defined by

$$(17) \qquad (x, y) \in R^{\cup} \quad \equiv \quad (y, x) \in R \quad .$$

Shunting of Functions Functions are very special sorts of relations because they enjoy a number of properties not generally enjoyed by relations proper. In point-free relation algebra, functions are characterized by a so-called *shunting* rule. The rule is that, relation f is a (total) *function* if and only if for all relations R and S,

$$f \bullet R \subseteq S \quad \equiv \quad R \subseteq f^{\cup} \bullet S \quad .$$

(Strictly we should specify the types of f, R and S. These details are omitted.) Note that, essentially by taking the converse of both sides of this equivalence, we also have, for all relations R and S (of appropriate type),

$$R \bullet f^{\cup} \subseteq S \quad \equiv \quad R \subseteq S \bullet f \quad .$$

An alternative, equivalent, characterisation of total functions is the following. The relation f of type $A \leftarrow B$ is a total function if and only if it is *simple*

$$f \bullet f^{\cup} \subseteq id_A$$

and *entire*

$$id_B \subseteq f^\cup \bullet f$$

where id_A and id_B denote the identity relations on the sets A and B, respectively. (It is easy to see that simplicity and entirety follow from the shunting rules. That the shunting rules follow from simplicity and entirety is an easy application of the monotonicity of relational composition.)

If f is a function then we denote by $f.y$ the unique value x such that $(x,y) \in f$. That is, for function f of type $A \sim B$ we have, for all x in A and all y in B,

$$(x,y) \in f \equiv x = f.y \quad .$$

(Given $y \in B$ the existence of $x \in A$ such that $(x,y) \in f$ is guaranteed by the entirety of f and its uniqueness is guaranteed by the simplicity of f.)

We now wish to define an arrow operator on pairs of relations. Suppose R and S are binary relations of type $A \sim B$ and $C \sim D$, respectively. Suppose that $(A, \sqsubseteq)$, $(B, \preceq)$, $(C, \trianglelefteq)$ and $(D, \leq)$ are partially ordered sets, then $R \leftarrow S$ is the binary relation of type

$$((A, \sqsubseteq) \leftarrow (C, \trianglelefteq)) \sim ((B, \preceq) \leftarrow (D, \leq))$$

defined by, for all (monotonic) functions $f \in (A, \sqsubseteq) \leftarrow (C, \trianglelefteq)$ and $g \in (B, \preceq) \leftarrow (D, \leq)$,

$$(18) \qquad (f,g) \in R \leftarrow S \quad \equiv \quad R \bullet g \supseteq f \bullet S \quad .$$

(The advantage of this formulation over the pointwise definition (1) is that it eliminates the universal quantification as well as the bound variables u and v.)

Properties From this definition (and basic properties of relation algebra) we can easily derive the following algebraic properties of the $\leftarrow$ operator [Bac90].

$$(19) \qquad (R \leftarrow S)^\cup \quad = \quad R^\cup \leftarrow S^\cup \quad ,$$

$$(20) \qquad (R \bullet S) \leftarrow (T \bullet U) \quad \supseteq \quad (R \leftarrow T) \bullet (S \leftarrow U) \quad , \text{ and}$$

For all monotonic functions $f \in (A, \sqsubseteq) \leftarrow (C, \trianglelefteq)$ and $g \in (B, \preceq) \leftarrow (D, \leq)$,

$$(21) \qquad f \leftarrow g^\cup \in ((A, \sqsubseteq) \leftarrow (D, \leq)) \leftarrow ((B, \preceq) \leftarrow (C, \trianglelefteq)) \quad .$$

(So $f \leftarrow g^\cup$ is a monotonic function.) Moreover, for all $h \in (B, \preceq) \leftarrow (C, \trianglelefteq)$,

$$(22) \qquad (f \leftarrow g^\cup).h \quad = \quad f \bullet h \bullet g \quad , \text{ and}$$

for all relations R and S,

$$(23) \qquad (R \bullet f) \leftarrow (S \bullet g^\cup) \quad = \quad (R \leftarrow S) \bullet (f \leftarrow g^\cup) \quad .$$

We refer to (19), (20) and (23) as the *distributivity* properties of $\leftarrow$.

Note carefully that the middle property expresses an inclusion whereas the other two are equalities. To see that the inclusion cannot be strengthened to an equality, let $\bot\!\!\!\bot_{A,B}$ denote the empty relation on sets A and B (that is the empty subset of $A \times B$). Also, let $\top\!\!\!\top_{A,B}$ denote the universal relation on sets A and B (that is the set $A \times B$). Then it suffices to note that

$$\bot\!\!\!\bot_{A,B} \leftarrow \bot\!\!\!\bot_{C,D} \;=\; \top\!\!\!\top_{(A \leftarrow C),(B \leftarrow D)}$$

whereas, for any *non-empty* relation R on the sets C and D,

$$\bot\!\!\!\bot_{A,B} \leftarrow R \;=\; \bot\!\!\!\bot_{(A \leftarrow C),(B \leftarrow D)} \quad .$$

By choosing R and S to be *non-empty* relations whose composition $R \bullet S$ is *empty* we get that $(\bot\!\!\!\bot_{A,B} \leftarrow R) \bullet (\bot\!\!\!\bot_{B,C} \leftarrow S)$ is the empty relation. This is strictly less than $\bot\!\!\!\bot_{A,C} \leftarrow (R \bullet S)$ which is the universal relation. (Sets A, B and C have also to be non-empty, of course, but otherwise they can be chosen arbitrarily.)

The $\leftarrow$ arrow operator is also monotonic in its left argument and anti-monotonic in its right argument. That is

$$(24) \qquad R \leftarrow S \;\supseteq\; T \leftarrow U \quad \Leftarrow \quad R \supseteq T \wedge S \subseteq U \quad .$$

Finally, using id_X to denote the identity relation on set X, we have:

$$(25) \qquad \mathsf{id}_A \leftarrow \mathsf{id}_B \;=\; \mathsf{id}_{A \leftarrow B} \quad .$$

Pointwise Ordering Suppose $(A, \sqsubseteq)$ and $(B, \preceq)$ are posets. Consider the monotonic functions with range A and domain B. The relation $\sqsubseteq \leftarrow \preceq$ is an ordering relation on such functions.

Lemma 26 Let $(A, \sqsubseteq)$ and $(B, \preceq)$ be posets. Then $\sqsubseteq \leftarrow \preceq$ is a partial ordering on the monotonic functions of type $A \leftarrow B$.

Proof The monotonic functions of type $A \leftarrow B$ are exactly the functions for which $\sqsubseteq \leftarrow \preceq$ is reflexive. Transitivity follows from (20) and antisymmetry from reflexivity of $\leq$ and $\leq \leftarrow \preceq$.
□

A simple but crucial property is that $\sqsubseteq \leftarrow \preceq$ is the usual pointwise ordering relation on functions. That is, for all functions f and g in $(A, \sqsubseteq) \leftarrow (B, \preceq)$,

$$\langle \forall u,v :: f.u \sqsubseteq g.v \;\Leftarrow\; u \preceq v \rangle \;\equiv\; \langle \forall u :: f.u \sqsubseteq g.u \rangle \quad .$$

Equivalently, in point-free relation algebra,

$$(27) \qquad \sqsubseteq \leftarrow \preceq \;=\; \sqsubseteq \leftarrow \mathsf{id}_{\preceq} \quad .$$

where $\mathsf{id}_{\preceq}$ denotes the identity relation on the carrier of the poset ordering $\preceq$. See lemma 50 for the generalisation of this property to higher order types.

Galois Connections Expressed in point-free relation algebra, the pair algebra corresponding to the Galois connection (abs, con) between $(A, \sqsubseteq)$ and $(B, \preceq)$ is the relation $abs^{\cup} \bullet \sqsubseteq$. The fact that the functions are indeed Galois connected is expressed very succinctly by the property

$$abs^{\cup} \bullet \sqsubseteq \quad = \quad \preceq \bullet con \quad .$$

An alternative characterization of Galois connections is in terms of *cancellation* properties. The pair of functions (abs, con) between $(A, \sqsubseteq)$ and $(B, \preceq)$ is Galois connected if they are both monotonic and, for all x in B and all y in A, $x \preceq con.(abs.x)$ and $abs.(con.y) \sqsubseteq y$. Expressed in point-free relation algebra, these are the rules:

$$con \bullet \sqsubseteq \quad \subseteq \quad \preceq \bullet con \quad ,$$

$$abs \bullet \preceq \quad \subseteq \quad \sqsubseteq \bullet abs \quad ,$$

$$\preceq \quad \subseteq \quad \preceq \bullet con \bullet abs$$

and

$$abs \bullet con \bullet \sqsubseteq \quad \subseteq \quad \sqsubseteq \quad .$$

The first two properties are the monotonicity properties, and the second two the cancellation properties.

7.1 Basic Constructions

The following lemma generalises what is known in the literature as the "independent attribute method" [NNH98, page 247]. Specifically, the independent attribute method is the case that the relator F is the product relator. For zero-ary relators the lemma is that, if $(A, \leq)$ is a poset, the pair (id_A, id_A) is a Galois connection between $(A, \leq)$ and itself. (This important result is undoubtedly well-known but often omitted in the literature.) We state and prove the lemma only for a unary relator. The extension to non-unary relators is easy but requires extra apparatus. (Replace posets, relations and functions everywhere by vectors of posets, relations and functions.)

Lemma 28 Let $(A, \leq)$ be a posets. Suppose F is a relator that preserves binary intersections. (That is, for all relations R and S, $F.(R \cap S) = F.R \cap F.S$.) Then $(F.A, F.\leq)$ is a poset.

Proof It is easy to verify that transitivity and reflexivity of $F.\leq$ follows from transitivity and reflexivity of $\leq$. Antisymmetry of $F.\leq$ is the property that

$$F.\leq \cap (F.\leq)^{\cup} \subseteq F.id_A \quad .$$

This follows from the antisymmetry of $\leq$ ($\leq \cap \leq^{\cup} \subseteq id_A$) and that F commutes with converse and the assumption that F preserves binary intersections.
□

Lemma 29 If f is a total function of type $A \leftarrow B$ and g is a total function of type $C \leftarrow D$ then $f \leftarrow g^{\cup}$ is a total function of type $(A \leftarrow D) \leftarrow (B \leftarrow C)$.

Proof Straightforward expansion of the definition of the arrow operator (18).
□

The following three properties will be used frequently; they are all easily proved by induction on the structure of type expressions. For all type expressions t and all assignments R and S of type variables to relations,

(30) $\quad R_t \;=\; [R,R]_t \quad ,$

(31) $\quad (R_t)^{\cup} \;=\; (R^{\cup})_t \quad$, and

(32) $\quad ([R,S]_t)^{\cup} \;=\; \left[R^{\cup}, S^{\cup}\right]_t \quad .$

Lemma 33 Suppose f and g are type assignments such that, for all type variables a, f_a and g_a are total functions of type $A_a \leftarrow B_a$ and $C_a \leftarrow D_a$, respectively. Then, for all type expressions t, $[f, g^{\cup}]_t$ is a total function of type $[A,D]_t \leftarrow [B,C]_t$.

Proof The proof is by induction on the structure of type expressions. First, if t is a type variable a, then $[f, g^{\cup}]_t = f_a$ which is a total function of type $A_a \leftarrow B_a$ (and hence of type $[A,D]_a \leftarrow [B,C]_a$).

Second, consider the case where $t = u \leftarrow v$. Then

$$\begin{array}{ll} & [f, g^{\cup}]_{u \leftarrow v} \\ = & \quad \{ \quad \text{definition: (7)} \quad \} \\ & [f, g^{\cup}]_u \leftarrow [g^{\cup}, f]_v \\ = & \quad \{ \quad (32) \quad \} \\ & [f, g^{\cup}]_u \leftarrow [g, f^{\cup}]_v^{\cup} \quad . \end{array}$$

But by the induction hypothesis and lemma 29, the above is a total function of type

$$([A,D]_u \leftarrow [D,A]_v) \leftarrow ([B,C]_u \leftarrow [C,B]_v) \quad .$$

That is, it is a total function of type $[A,D]_t \leftarrow [B,C]_t$.

If t is a relator application then, by definition,

$$[f\,,g^{\cup}]_{F.(u\ldots\ v)} \;=\; F.([f\,,g^{\cup}]_u \ldots\ [f\,,g^{\cup}]_v) \quad .$$

Relators preserve total functions, so by the induction hypothesis this is a total function. Also, its type is

$$F.([A,D]_u \ldots\ [A,D]_v) \leftarrow F.([B,C]_u \ldots\ [B,C]_v)$$

which, because F is a functor is the same as

$$[A,D]_{F.(u\ldots\ v)} \leftarrow [B,C]_{F.(u\ldots\ v)}$$

as required.
□

Lemma 34 For all assignments f, g, h and k of functions to type variables and for all type expressions t

$$[f\,,g^{\cup}]_t \bullet [h\,,k^{\cup}]_t \;=\; [f\bullet h\,,(k\bullet g)^{\cup}]_t \quad .$$

(Note the contravariance in the second argument.)

Proof The induction hypothesis is the equality above together with the symmetric equality

$$[k\,,h^{\cup}]_t \bullet [g\,,f^{\cup}]_t \;=\; [k\bullet g\,,(f\bullet h)^{\cup}]_t \quad .$$

The basis follows immediately from (6).

For the case $t = u \leftarrow v$, we have:

$$\begin{array}{ll}
 & [f\,,g^{\cup}]_t \bullet [h\,,k^{\cup}]_t \\
= & \quad \{ \quad t = u \leftarrow v\text{, (7) and (11)} \quad \} \\
 & ([f\,,g^{\cup}]_u \leftarrow [g\,,f^{\cup}]_v^{\cup}) \bullet ([h\,,k^{\cup}]_u \leftarrow [k\,,h^{\cup}]_v^{\cup}) \\
= & \quad \{ \quad \text{(23) — which is applicable because} \\
 & \qquad\quad [h\,,k^{\cup}]_u \text{ and } [k\,,h^{\cup}]_v \text{ are functions} \quad \} \\
 & [f\,,g^{\cup}]_u \bullet [h\,,k^{\cup}]_u \leftarrow [g\,,f^{\cup}]_v^{\cup} \bullet [k\,,h^{\cup}]_v^{\cup} \\
= & \quad \{ \quad \text{converse} \quad \} \\
 & [f\,,g^{\cup}]_u \bullet [h\,,k^{\cup}]_u \leftarrow ([k\,,h^{\cup}]_v \bullet [g\,,f^{\cup}]_v)^{\cup} \\
= & \quad \{ \quad \text{induction hypothesis} \quad \} \\
 & [f\bullet h\,,(k\bullet g)^{\cup}]_u \leftarrow [k\bullet g\,,(f\bullet h)^{\cup}]_v^{\cup} \\
= & \quad \{ \quad \text{(11), (22) and } t = u \leftarrow v \quad \} \\
 & [f\bullet h\,,(k\bullet g)^{\cup}]_t \quad .
\end{array}$$

The proof of the second equality is completely symmetrical.

In the case that t is a relator application the claimed equalities are immediate consequences of the fact that relators distribute through composition together with (8).
□

Lemma 35 (Monotonicity) Suppose R, S, T and U are type assignments such that for all type variables a:

$$R_a \subseteq T_a \;\wedge\; S_a \supseteq U_a \quad .$$

Then, for all type expressions t,

$$[R,S]_t \;\subseteq\; [T,U]_t \quad .$$

Proof Straightforward induction on the structure of type expressions using (30) for the base case, (7) and (24) for the case that $t = u \leftarrow v$, and the monotonicity of relators and (8) for the remaining case.
□

Lemma 36 Let t be a type expression and let $\sqsubseteq$ and $\preceq$ assign partial ordering relations to the type variables in t. Let A and B be the corresponding poset assignments, respectively. (So for each type variable a, $\sqsubseteq_a$ is a partial ordering relation on A_a and $\preceq_a$ is a partial ordering relation on B_a.) Then $[\sqsubseteq, \preceq]_t$ is a partial ordering on $[A, B]_t$. In particular, $\sqsubseteq_t$ is a partial ordering relation on A_t.

Proof Straightforward induction on the stucture of type expressions using lemmas 26 and 28. ((30) is then used to derive the special case.)
□

Lemma 37 (Factorisation of functions) Suppose R, S, f and g are type assignments such that, for all type variables a, f_a and g_a are total functions and the compositions $R_a \bullet f_a$ and $S_a \bullet g_a^\cup$ are well defined. Then, for all type expressions t,

(38) $$[R,S]_t \bullet \left[f, g^\cup\right]_t \;\supseteq\; \left[R \bullet f \;,\; S \bullet g^\cup\right]_t \quad \text{, and}$$

(39) $$[S,R]_t \bullet \left[g^\cup, f\right]_t \;\subseteq\; \left[S \bullet g^\cup \;,\; R \bullet f\right]_t \quad .$$

Proof The proof is by induction on the structure of type expressions. First, consider the case that t is a type variable a. By applying property (6), we have

$$[R,S]_a \bullet [f, g^\cup]_a \;=\; R_a \bullet f_a \;=\; [R \bullet f \;,\; S \bullet g^\cup]_a \quad \text{, and}$$

$$[S,R]_a \bullet [g^\cup, f]_a \;=\; S_a \bullet g^\cup_a \;=\; [S \bullet g^\cup \;,\; R \bullet f]_a \quad .$$

Second, consider the case where $t = u \leftarrow v$. For property (38) we reason:

$$\begin{array}{ll}
 & [R,S]_{u \leftarrow v} \bullet [f, g^\cup]_{u \leftarrow v} \\
= & \quad \{ \quad \text{definition: (7)} \quad \} \\
 & ([R,S]_u \leftarrow [S,R]_v) \bullet ([f, g^\cup]_u \leftarrow [g^\cup, f]_v) \\
= & \quad \{ \quad \text{distributivity: (23), lemma 33 and (32)} \quad \} \\
 & ([R,S]_u \bullet [f, g^\cup]_u) \leftarrow ([S,R]_v \bullet [g^\cup, f]_v) \\
\supseteq & \quad \{ \quad \text{monotonicity: (24), induction hypotheses: (38) and (39)} \quad \} \\
 & [R \bullet f \;,\; S \bullet g^\cup]_u \;\leftarrow\; [S \bullet g^\cup \;,\; R \bullet f]_v \\
= & \quad \{ \quad \text{definition: (7)} \quad \} \\
 & [R \bullet f \;,\; S \bullet g^\cup]_{u \leftarrow v} \quad .
\end{array}$$

For property (39) we reason:

$$\begin{array}{ll}
 & [S,R]_{u \leftarrow v} \bullet [g^\cup, f]_{u \leftarrow v} \\
= & \quad \{ \quad \text{definition: (7)} \quad \} \\
 & ([S,R]_u \leftarrow [R,S]_v) \bullet ([g^\cup, f]_u \leftarrow [f, g^\cup]_v) \\
\subseteq & \quad \{ \quad \text{distributivity: (20)} \quad \} \\
 & ([S,R]_u \bullet [g^\cup, f]_u) \leftarrow ([R,S]_v \bullet [f, g^\cup]_v) \\
\subseteq & \quad \{ \quad \text{monotonicity: (24), induction hypotheses} \quad \} \\
 & [S \bullet g^\cup \;,\; R \bullet f]_u \;\leftarrow\; [R \bullet f \;,\; S \bullet g^\cup]_v \\
= & \quad \{ \quad \text{definition: (7)} \quad \} \\
 & [S \bullet g^\cup \;,\; R \bullet f]_{u \leftarrow v} \quad .
\end{array}$$

The case where t is a relator application is straightforward and thus omitted.
□

7.2 The Theorems

Lemma 33 gives the type of the function that maps a pair of assignments f, g to the assignment $[f, g^\cup]$. This function is "natural" if types can be replaced by relations and the property still holds. This is what the next theorem is about.

Theorem 40 (Naturality of higher order function assignments) Suppose that f, g, h and k assign total functions to type variables, and R and S assign relations to type variables. Suppose further that, for all type variables a, $(f_a,h_a) \in R_a \leftarrow S_a$ and $(g_a,k_a) \in T_a \leftarrow U_a$. Then, for all type expressions t,

$$([f,g^\cup]_t \ , \ [h,k^\cup]_t) \in [R,U]_t \leftarrow [S,T]_t \quad .$$

Proof

$$\begin{array}{ll}
& ([f,g^\cup]_t \ , \ [h,k^\cup]_t) \in [R,U]_t \leftarrow [S,T]_t \\
= & \quad \{ \quad \text{definition of } \leftarrow \ (18) \quad \} \\
& [R,U]_t \bullet [h,k^\cup]_t \ \supseteq \ [f,g^\cup]_t \bullet [S,T]_t \\
\Leftarrow & \quad \{ \quad (38), (39) \text{ together with the fact that} \\
& \qquad \text{converse is a poset isomorphism, transitivity of } \supseteq \quad \} \\
& [R \bullet h \ , \ U \bullet k^\cup]_t \ \supseteq \ [f \bullet S \ , \ g^\cup \bullet T]_t \\
\Leftarrow & \quad \{ \quad \text{monotonicity: (35)} \quad \} \\
& \langle \forall a \ :: \ (R \bullet h)_a \ \supseteq \ (f \bullet S)_a \ \wedge \ (U \bullet k^\cup)_a \subseteq (g^\cup \bullet T)_a \rangle \\
= & \quad \{ \quad \text{definition: (3)} \quad \} \\
& \langle \forall a \ :: \ R_a \bullet h_a \ \supseteq \ f_a \bullet S_a \ \wedge \ U_a \bullet k_a^\cup \subseteq g_a^\cup \bullet T_a \rangle \\
= & \quad \{ \quad \text{shunting of functions and definition of } \leftarrow \ (18) \quad \} \\
& \langle \forall a \ :: \ (f_a,h_a) \in R_a \leftarrow S_a \ \wedge \ (g_a,k_a) \in T_a \leftarrow U_a \rangle \quad .
\end{array}$$

□

Theorem 40 has a number of corollaries. The following is the lemma proved by Backhouse in [Bac90].

Corollary 41 Suppose f and g assign total functions to type variables, where f_a and g_a have type $A_a \leftarrow B_a$ and $C_a \leftarrow D_a$, respectively. Then, for all type expressions t,

$$[f,C]_t \bullet [B \ , g^\cup]_t^\cup \ \subseteq \ [f,g]_t \ \subseteq \ [A \ , g^\cup]_t^\cup \bullet [f,D]_t \quad .$$

Proof We observe that $(f_a,B_a) \in f_a \leftarrow B_a$ and $(C_a,g_a) \in C_a \leftarrow g_a$ and use this observation to instantiate theorem 40. Specifically,

$$\begin{array}{ll}
& true \\
= & \quad \{ \quad \text{theorem 40 with } f,g,h,k := f,C,B,g \\
& \qquad \text{and } R,S,T,U := f,B,C,g \quad \}
\end{array}$$

$$
\begin{array}{ll}
 & ([f\,,C^{\cup}]_t\ ,\ [B\,,g^{\cup}]_t) \in [f,g]_t \leftarrow [B,C]_t \\
= & \quad \{ \quad \text{definition of arrow (18)} \quad \} \\
 & [f,g]_t \bullet [B\,,g^{\cup}]_t \ \supseteq\ [f\,,C^{\cup}]_t \bullet [B,C]_t \\
= & \quad \{ \quad [B,C]_t \text{ is the identity function, shunting of functions,} \\
 & \qquad C = C^{\cup} \quad \} \\
 & [f,C]_t \bullet [B\,,g^{\cup}]_t^{\cup} \ \subseteq\ [f,g]_t \quad .
\end{array}
$$

The second inclusion is obtained similarly by observing that $(A_a,f_a) \in A_a \leftarrow f_a$ and $(g_a,D_a) \in g_a \leftarrow D_a$.
□

Corollary 42 (Dinaturality) Suppose θ is a parametrically polymorphic function of type $t = u \leftarrow v$ for all instances of the type variables in t. Then θ is "dinatural". Specifically, for all assignments f of total functions to the type variables, where f_a has type $A_a \leftarrow B_a$,

$$[A\,,f^{\cup}]_u \bullet \theta(A) \bullet [f,A]_v \ =\ [f,B]_u \bullet \theta(B) \bullet [B\,,f^{\cup}]_v \quad .$$

Proof We combine the free theorem with corollary 41.

$$
\begin{array}{ll}
 & \text{true} \\
= & \quad \{ \quad \text{free theorem} \quad \} \\
 & (\theta(A)\,,\ \theta(B)) \in f_t \\
\Rightarrow & \quad \{ \quad f_t \ =\ [f,f]_t \text{, corollary 41} \quad \} \\
 & (\theta(A)\,,\ \theta(B)) \in [A\,,f^{\cup}]_t^{\cup} \bullet [f,B]_t \\
= & \quad \{ \quad [A\,,f^{\cup}]_t \text{ and } [f,B]_t \text{ are functions} \quad \} \\
 & [A\,,f^{\cup}]_t\,.\theta(A) \ =\ [f,B]_t\,.\theta(B) \\
= & \quad \{ \quad t = u \leftarrow v \text{, definition (7)} \quad \} \\
 & ([A\,,f^{\cup}]_u \leftarrow [f^{\cup}\,,A]_v).\theta(A) \ =\ ([f,B]_u \leftarrow [B,f]_v).\theta(B) \\
= & \quad \{ \quad \text{converse (11) and definition (22)} \quad \} \\
 & [A\,,f^{\cup}]_u \bullet \theta(A) \bullet [f,A]_v \ =\ [f,B]_u \bullet \theta(B) \bullet [B\,,f^{\cup}]_v \quad .
\end{array}
$$

□

An example of the dinaturality property is provided by function composition. Composition has type

$$(a \leftarrow b) \leftarrow (a \leftarrow c) \leftarrow (c \leftarrow b)$$

for all instances of the type variables a, b and c. Suppose a function denoted by the infix operator "$\circ$" has the same type as composition. Then, if we expand the dinaturality property of this function we get that, for all functions f, g, h, k and m,

$$f \bullet ((g \bullet h) \circ (k \bullet m)) \;=\; ((f \bullet g) \circ (h \bullet k)) \bullet m \quad .$$

(We leave this to the reader to verify. The different instances of "$\circ$" have not been distinguished for the purposes of readability.) In particular, by instantiating g and k with identity functions we get that

$$f \bullet (h \circ m) \;=\; (f \circ h) \bullet m \quad .$$

In words, any function that has the same type as function composition must "associate" with function composition. Of course, composition has the same type as itself, so the dinaturality property predicts that function composition is associative. (Note that associativity was not used in deriving this special case of the dinaturality property. Otherwise the argument would be circular!) Moreover, by now taking m to be identity function (denoted here by id) we get

$$f \bullet h \;=\; f \circ h \;\Leftarrow\; h \circ id = h \quad .$$

From this we conclude that function composition is uniquely defined by the fact that it is parametrically polymorphic and has the identity function as unit.

Corollary 43 Suppose that f and g assign total functions to type variables, and R and S assign relations to type variables. Suppose further that, for all type variables a, $f_a \in R_a \leftarrow S_a$ and $g_a \in S_a \leftarrow R_a$. Then, for all type expressions t,

$$[f, g^{\cup}]_t \in R_t \leftarrow S_t \quad \text{, and}$$

$$[g, f^{\cup}]_t \in S_t \leftarrow R_t \quad .$$

Thus, if, for each a, R_a and S_a are ordering relations, and f_a and g_a are monotonic functions, then $[f, g^{\cup}]_t$ and $[g, f^{\cup}]_t$ are monotonic functions.

Proof Make the instantiation $h,k,T,U := f,g,S,R$ in lemma 40 and use (30).
□

Lemma 44 Suppose that f and g assign total functions to type variables, and R and S assign relations to type variables. Suppose further that, for all type variables a, the following cancellation property holds:

$$g_a \bullet R_a \subseteq f_a^\cup \bullet R_a \quad .$$

Then, for all type expressions t,

$$[g, f^\cup]_t \bullet R_t \subseteq (f^\cup \bullet R)_t \subseteq [f^\cup, g]_t \bullet R_t \quad .$$

Proof

$$\begin{array}{ll} & [g, f^\cup]_t \bullet R_t \subseteq (f^\cup \bullet R)_t \subseteq [f^\cup, g]_t \bullet R_t \\ \Leftarrow & \quad \{ \quad \text{(30), (38), (39) together with the fact that} \\ & \qquad \text{converse is a poset isomorphism, transitivity of } \supseteq \quad \} \\ & [g \bullet R, f^\cup \bullet R]_t \subseteq [f^\cup \bullet R, f^\cup \bullet R]_t \subseteq [f^\cup \bullet R, g \bullet R]_t \\ \Leftarrow & \quad \{ \quad \text{monotonicity: (35)} \quad \} \\ & \langle \forall a :: g_a \bullet R_a \subseteq f_a^\cup \bullet R_a \subseteq f_a^\cup \bullet R_a \ \wedge \ f_a^\cup \bullet R_a \supseteq f_a^\cup \bullet R_a \supseteq g_a \bullet R_a \rangle \\ = & \quad \{ \quad \text{reflexivity of } \subseteq \text{ and assumption: } g_a \bullet R_a \subseteq f_a^\cup \bullet R_a \quad \} \\ & true \quad . \end{array}$$

□

Lemma 45 Suppose that f and g assign total functions to type variables, and R and S assign relations to type variables. Suppose further that, for all type variables a, the following cancellation property holds:

$$S_a \bullet f_a^\cup \subseteq S_a \bullet g_a \quad .$$

Then, for all type expressions t,

$$S_t \bullet [f^\cup, g]_t \subseteq (S \bullet g)_t \subseteq S_t \bullet [g, f^\cup]_t \quad .$$

Proof Similar to the proof of theorem 44.
□

7.3 Additions

Suppose V and W are two variable assignments of the same kind. Then the variable assignment $[\![V,W]\!]$ is defined inductively as follows:

(46) $$[\![V,W]\!]_a = V_a \quad ,$$

(47) $$[\![V,W]\!]_{u \leftarrow v} = [\![V,W]\!]_u \leftarrow W_v \quad ,$$

(48) $$[\![V,W]\!]_{F.(u \ldots v)} = F.([\![V,W]\!]_u \ldots [\![V,W]\!]_v) \quad .$$

The assignment $[\![V,W]\!]$ applies the assignment V to the highest positive occurrences of the variables and applies assignment W to all others. For example,

$$[\![(a,b := \sqsubseteq,\trianglelefteq)\ ,\ (a,b := I,J)]\!]_{\mathrm{List}.((a\leftarrow b)\times b)\ \leftarrow\ a}$$
$$=\ \mathrm{List}.((\sqsubseteq\leftarrow J)\times\trianglelefteq)\leftarrow I\quad .$$

If I and J are identity relations on posets A and B, respectively, $\mathrm{List}.((\sqsubseteq\leftarrow J)\times\trianglelefteq)\leftarrow I$ is the pointwise ordering relation on functions of type $\mathrm{List}.((A\leftarrow B)\times B)\leftarrow B$. As the next lemma states, it is equal to the relation

$$\mathrm{List}.((\sqsubseteq\leftarrow\trianglelefteq)\times\trianglelefteq)\leftarrow\sqsubseteq\quad .$$

Lemma 49 For all partial orderings $\sqsubseteq$ and $\preceq$ and all functions g,

$$\sqsubseteq\leftarrow\preceq\bullet g\ =\ \sqsubseteq\leftarrow g\quad\text{and}$$

$$\sqsubseteq\leftarrow g^{\cup}\bullet\preceq\ =\ \sqsubseteq\leftarrow g^{\cup}\quad .$$

Proof We have, for all monotonic functions h and k,

$$\begin{array}{ll}
 & (h,k)\in\sqsubseteq\leftarrow\preceq\bullet g\\
= & \quad\{\quad\text{definitions: (1), (16)}\quad\}\\
 & \langle\forall u,v\ ::\ h.u\sqsubseteq k.v\ \Leftarrow\ u\preceq g.v\rangle\\
= & \quad\{\quad(\Rightarrow)\ \text{reflexivity of}\ \preceq\\
 & \qquad\quad(\Leftarrow)\ h\ \text{is monotonic, transitivity of}\ \sqsubseteq\quad\}\\
 & \langle\forall v\ ::\ h.(g.v)\sqsubseteq k.v\rangle\\
= & \quad\{\quad\text{one-point rule}\quad\}\\
 & \langle\forall u,v\ ::\ h.u\sqsubseteq k.v\ \Leftarrow\ u=g.v\rangle\\
= & \quad\{\quad\text{definitions: (1), (16)}\quad\}\\
 & (h,k)\in\sqsubseteq\leftarrow g\quad .
\end{array}$$

The second claim is proved similarly. (This is where monotonicity of k is used.)
□

Lemma 49 is used frequently below, often in combination with the distributivity property (23). Its most immediate application is property (27). Generalising to higher order types we get the following lemma.

Lemma 50 (Higher Order Pointwise Orderings) If for each variable a, $\sqsubseteq_a$ is a partial ordering relation on the set A_a and id_a is the identity relation on A_a then, for all type expressions t,

$$[\![\sqsubseteq,id]\!]_t\ =\ \sqsubseteq_t\quad .$$

Proof By induction on type expressions. The basis is a trivial use of (46). For $t = u \leftarrow v$ we apply the property (27):

$$
\begin{array}{ll}
 & \sqsubseteq_{u \leftarrow v} \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \sqsubseteq_u \leftarrow \sqsubseteq_v \\
= & \quad \{ \quad \text{lemma 49} \quad \} \\
 & \sqsubseteq_u \leftarrow id_v \\
= & \quad \{ \quad \text{induction hypothesis} \quad \} \\
 & [\![\sqsubseteq , id]\!]_u \leftarrow id_v \\
= & \quad \{ \quad t = u \leftarrow v \text{, definition: (47)} \quad \} \\
 & [\![\sqsubseteq , id]\!]_t \quad .
\end{array}
$$

The case $t = F.(u \ldots v)$ is straightforward:

$$
\begin{array}{ll}
 & \sqsubseteq_{F.(u \ldots v)} \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & F.(\sqsubseteq_u \ldots \sqsubseteq_v) \\
= & \quad \{ \quad \text{induction hypothesis} \quad \} \\
 & F.([\![\sqsubseteq , id]\!]_u \ldots [\![\sqsubseteq , id]\!]_v) \\
= & \quad \{ \quad t = F.(u \ldots v) \text{, definition: (48)} \quad \} \\
 & [\![\sqsubseteq , id]\!]_t \quad .
\end{array}
$$

□

In words, lemma 50 states that the pointwise ordering of functions, $[\![\sqsubseteq , id]\!]_t$, is the logical relation $\sqsubseteq_t$ generated by the given base orderings.

The next lemma can be seen as a special case of Abramsky's "uniformization theorem" [Abr90, proposition 6.4]. (Abramsky does not assume that the function g has an upper adjoint.)

Lemma 51 (Uniformization) If f is a monotonic function with range $(A , \sqsubseteq)$ and the pair $(g , g^\sharp)$ is a Galois connection between the posets $(B , \preceq)$ and $(C , \leq)$ then

$$f^\cup \bullet \sqsubseteq \leftarrow g^\cup \bullet \preceq \quad = \quad (f^\cup \leftarrow g^\sharp) \bullet (\sqsubseteq \leftarrow \preceq) \quad .$$

Also,

$$\sqsubseteq \bullet f \leftarrow \leq \bullet g^\sharp \quad = \quad (\sqsubseteq \leftarrow \leq) \bullet (f \leftarrow g^\cup) \quad .$$

Proof

$$
\begin{array}{ll}
 & f^{\cup} \bullet \sqsubseteq \leftarrow g^{\cup} \bullet \preceq \\
= & \quad \{ \quad (g\,,\, g^{\sharp}) \text{ is a Galois connection.} \\
 & \qquad \text{So, } g^{\cup} \bullet \preceq \;=\; \leq \bullet g^{\sharp}\,. \quad \} \\
 & f^{\cup} \bullet \sqsubseteq \leftarrow \leq \bullet g^{\sharp} \\
= & \quad \{ \quad \text{distributivity (23) and lemma 49} \quad \} \\
 & (f^{\cup} \leftarrow \mathrm{id}_C) \bullet (\sqsubseteq \leftarrow g^{\sharp}) \\
= & \quad \{ \quad \text{distributivity: (23)} \quad \} \\
 & (f^{\cup} \leftarrow g^{\sharp}) \bullet (\sqsubseteq \leftarrow \mathrm{id}_B) \\
= & \quad \{ \quad \text{lemma 49} \quad \} \\
 & (f^{\cup} \leftarrow g^{\sharp}) \bullet (\sqsubseteq \leftarrow \preceq) \quad .
\end{array}
$$

The second property is proved similarly.
□

Lemma 52 Suppose we are given two assignments of Galois connections $(f_a\,,g_a)$ between the posets $(A_a\,,\sqsubseteq_a)$ and $(B_a\,,\preceq_a)$ and $(h_a\,,k_a)$ between the posets $(C_a\,,\trianglelefteq_a)$ and $(D_a\,,\leq_a)$, for each type variable a. Then, for all type expressions t,

$$[f\,,k^{\cup}]_t^{\cup} \bullet [\sqsubseteq\,,\trianglelefteq]_t \;=\; [\preceq\,,\leq]_t \bullet [g\,,h^{\cup}]_t \quad .$$

In particular, the pair of functions $([f\,,k^{\cup}]_t\;,\;[g\,,h^{\cup}]_t)$ forms a Galois connection between the partially ordered sets $([A\,,C]_t\;,\;[\sqsubseteq\,,\trianglelefteq]_t)$ and $([B\,,D]_t\;,\;[\preceq\,,\leq]_t)$.

Proof Before embarking on the proof, we remark that lemma 33 establishes that $[f\,,k^{\cup}]_t$ and $[g\;,\;h^{\cup}]_t$ are functions (of the right type) and lemma 36 that $[\sqsubseteq\,,\trianglelefteq]_t$ and $[\preceq\,,\leq]_t$ are partial ordering relations. So it suffices only to prove the equality. This we do as follows.

The induction hypothesis is the equality stated above, together with the symmetric equality:

$$[h\,,g^{\cup}]_t^{\cup} \bullet [\trianglelefteq\,,\sqsubseteq]_t \;=\; [\leq\,,\preceq]_t \bullet [k\,,f^{\cup}]_t \quad .$$

We are given the equalities: for all type variables a,

$$f_a^{\cup} \bullet \sqsubseteq_a \;=\; \preceq_a \bullet g_a \quad \text{, and}$$

$$h_a^{\cup} \bullet \trianglelefteq_a \;=\; \leq_a \bullet k_a \quad .$$

Combined with (30), these establish the basis of the proof.

For the case $t = u \leftarrow v$ we have:

$$
\begin{array}{ll}
 & [f, k^{\cup}]^{\cup}_{t} \bullet [\sqsubseteq, \trianglelefteq]_{t} \\
= & \quad \{ \quad t = u \leftarrow v \text{, definitions (4) and (7)} \quad \} \\
 & ([f, k^{\cup}]_{u} \leftarrow [k^{\cup}, f]_{v})^{\cup} \bullet ([\sqsubseteq, \trianglelefteq]_{u} \leftarrow [\trianglelefteq, \sqsubseteq]_{v}) \\
= & \quad \{ \quad \text{converse: (19) and (11)} \quad \} \\
 & ([f, k^{\cup}]^{\cup}_{u} \leftarrow [k, f^{\cup}]_{v}) \bullet ([\sqsubseteq, \trianglelefteq]_{u} \leftarrow [\trianglelefteq, \sqsubseteq]_{v}) \\
= & \quad \{ \quad \text{uniformization (lemma 51)} \\
 & \qquad \text{and induction hypothesis (specifically that } [k, f^{\cup}]_{v} \text{ is} \\
 & \qquad \text{the upper adjoint of } [h, g^{\cup}]_{v} \text{)} \quad \} \\
 & [f, k^{\cup}]^{\cup}_{u} \bullet [\sqsubseteq, \trianglelefteq]_{u} \ \leftarrow \ [h, g^{\cup}]^{\cup}_{v} \bullet [\trianglelefteq, \sqsubseteq]_{v} \\
= & \quad \{ \quad \text{induction hypothesis} \quad \} \\
 & [\preceq, \leq]_{u} \bullet [g, h^{\cup}]_{u} \ \leftarrow \ [\leq, \preceq]_{v} \bullet [k, f^{\cup}]_{v} \\
= & \quad \{ \quad \text{uniformization (lemma 51)} \\
 & \qquad \text{and induction hypothesis (specifically that } [k, f^{\cup}]_{v} \text{ is} \\
 & \qquad \text{the upper adjoint of } [h, g^{\cup}]_{v} \text{)} \quad \} \\
 & ([\preceq, \leq]_{u} \leftarrow [\leq, \preceq]_{v}) \bullet ([g, h^{\cup}]_{u} \leftarrow [h, g^{\cup}]^{\cup}_{v}) \\
= & \quad \{ \quad \text{converse (11), } t = u \leftarrow v \text{, definitions (4) and (7)} \quad \} \\
 & [\preceq, \leq]_{t} \bullet [g, h^{\cup}]_{t} \quad .
\end{array}
$$

The proof of the second equality is completely symmetric.

Finally, the case $t = F.(u \ldots v)$ is a straightforward application of the fact that relators distribute through composition and commute with converse.

□

Lemma 53 If the base Galois connections defined in lemma 52 are all perfect then so are the constructed Galois connections.

Proof That the base connections are perfect means that $f_a \bullet g_a = A_a$ and $h_a \bullet k_a = C_a$. (We are taking the liberty of using A_a and C_a to denote identity relations here.) We have to prove that, for all type expressions t,

$$[f, k^{\cup}]_{t} \bullet [g, h^{\cup}]_{t} \ = \ [A, C]_{t} \quad .$$

The proof is an application of lemma 34:

$$
\begin{array}{ll}
 & [f,k^\cup]_t \bullet [g,h^\cup]_t \;=\; [A,C]_t \\
= & \quad \{ \quad \text{lemma 34} \quad \} \\
 & [f \bullet g, (h \bullet k)^\cup]_t \;=\; [A,C]_t \\
\Leftarrow & \quad \{ \quad \text{definition, converse of} \quad \} \\
 & \langle \forall a :: (f \bullet g)_a = A_a \;\wedge\; (h \bullet k)^\cup_a = C_a \rangle \\
= & \quad \{ \quad \text{(3), assumption, } C^\cup_a = C_a \quad \} \\
 & true \quad .
\end{array}
$$

□

In the following theorems we suppose we are given an assignment of Galois connections (abs_a, con_a) between the posets $(A_a, \sqsubseteq_a)$ and $(B_a, \preceq_a)$ for each type variable a. In other words, we are given six functions A, B, $\sqsubseteq$, $\preceq$, abs and con on the type variables. For each variable a, we define P_a to be the pair algebra corresponding to the given Galois connection. That is,

(54) $\quad abs^\cup_a \bullet \sqsubseteq_a \;=\; P_a \;=\; \preceq_a \bullet con_a \quad .$

Theorem 55 (Logical Pair Algebras) For all type expressions t,

$$[abs, con^\cup]^\cup_t \bullet \sqsubseteq_t \;=\; P_t \;=\; \preceq_t \bullet [con, abs^\cup]_t \quad .$$

In words, the logical relation P defines a Galois connection for all type expressions t, namely the pair of functions $([abs, con^\cup]_t \;,\; [con, abs^\cup]_t)$ connecting the partially ordered sets $(A_t, \sqsubseteq_t)$ and $(B_t, \preceq_t)$. Moreover, P_t is a pair algebra for all type expressions t.

Proof The proof is by induction on the structure of type expressions. The basis is the assumption (54) combined with (6) and (3). For $t = u \leftarrow v$ we have:

$$
\begin{array}{ll}
 & [abs, con^\cup]^\cup_t \bullet \sqsubseteq_t \\
= & \quad \{ \quad t = u \leftarrow v \text{, definitions (4) and (7)} \quad \} \\
 & ([abs, con^\cup]_u \leftarrow [con^\cup, abs]_v)^\cup \bullet (\sqsubseteq_u \leftarrow \sqsubseteq_v) \\
= & \quad \{ \quad \text{converse: (19) and (11)} \quad \} \\
 & ([abs, con^\cup]^\cup_u \leftarrow [con, abs^\cup]_v) \bullet (\sqsubseteq_u \leftarrow \sqsubseteq_v) \\
= & \quad \{ \quad \text{uniformization (lemma 51) and induction hypothesis} \quad \} \\
 & [abs, con^\cup]^\cup_u \bullet \sqsubseteq_u \;\leftarrow\; [abs, con^\cup]^\cup_v \bullet \sqsubseteq_v \\
= & \quad \{ \quad \text{induction hypothesis} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
 & P_u \leftarrow P_v \\
= & \quad \{ \quad t = u \leftarrow v \text{, definition (4)} \quad \} \\
 & P_t \quad .
\end{array}
$$

The proof of the second equality is completely symmetric.

Finally, the case $t = F.(u \ldots v)$ is a straightforward application of the fact that relators distribute through composition and commute with converse.

□

Lemma 56

$$P_t \;=\; [\![abs^\cup \bullet \sqsubseteq \,,\; [abs, con^\cup]^\cup]\!]_t \quad .$$

Proof The proof is by induction on the structure of type expressions. The basis is trivial. For $t = u \leftarrow v$ we have:

$$
\begin{array}{ll}
 & P_t \\
= & \quad \{ \quad t = u \leftarrow v \text{, definition (4)} \quad \} \\
 & P_u \leftarrow P_v \\
= & \quad \{ \quad \text{theorem 55} \quad \} \\
 & P_u \;\leftarrow\; [abs, con^\cup]^\cup_v \bullet \sqsubseteq_v \\
= & \quad \{ \quad \text{distributivity (23) and lemma 49} \\
 & \qquad \text{(noting that } P_u = [abs, con^\cup]^\cup_u \bullet \sqsubseteq_u \text{)} \quad \} \\
 & P_u \leftarrow [abs, con^\cup]^\cup_v \\
= & \quad \{ \quad \text{induction hypothesis} \quad \} \\
 & [\![abs^\cup \bullet \sqsubseteq \,,\; [abs, con^\cup]^\cup]\!]_u \;\leftarrow\; [abs, con^\cup]^\cup_v \\
= & \quad \{ \quad \text{definition: (47)} \quad \} \\
 & [\![abs^\cup \bullet \sqsubseteq \,,\; [abs, con^\cup]^\cup]\!]_{u \leftarrow v} \\
= & \quad \{ \quad t = u \leftarrow v \quad \} \\
 & [\![abs^\cup \bullet \sqsubseteq \,,\; [abs, con^\cup]^\cup]\!]_t \quad .
\end{array}
$$

□

Corollary 57 (Safety For Free) If θ has type $t = a \leftarrow v$, where a is a type variable and v is a type expression, and is parametrically polymorphic in all the type variables in t then

$$abs_a \bullet \theta(B) \quad \dot{\sqsubseteq} \quad \theta(A) \bullet [abs, con^{\cup}]_{\nu}$$

where $\dot{\sqsubseteq}$ denotes the pointwise ordering $[\![\sqsubseteq, A]\!]_t$ on functions with range A_a.

Proof The assumption that θ has type t and is parametrically polymorphic in all the type variables in $a \leftarrow \nu$ means that

$$(\theta(B), \theta(A)) \in P_{a \leftarrow \nu} \quad .$$

Now,

$$\begin{array}{ll} & P_{a \leftarrow \nu} \\ = & \quad \{ \quad \text{lemma 56} \quad \} \\ & [\![abs^{\cup} \bullet \sqsubseteq , \ [abs, con^{\cup}]^{\cup}]\!]_{a \leftarrow \nu} \\ = & \quad \{ \quad \text{definitions: (47) and (3)} \quad \} \\ & abs_a^{\cup} \bullet \sqsubseteq_a \leftarrow [abs, con^{\cup}]_{\nu}^{\cup} \\ = & \quad \{ \quad \text{distributivity: (23)} \quad \} \\ & (abs_a^{\cup} \leftarrow A_{\nu}) \bullet (\sqsubseteq_a \leftarrow A_{\nu}) \bullet (A_a \leftarrow [abs, con^{\cup}]_{\nu}^{\cup}) \\ = & \quad \{ \quad \dot{\sqsubseteq} = [\![\sqsubseteq, A]\!]_{a \leftarrow \nu} = \sqsubseteq_a \leftarrow A_{\nu}, \\ & \qquad \text{converse: (19)} \quad \} \\ & (abs_a \leftarrow A_{\nu})^{\cup} \bullet \dot{\sqsubseteq} \bullet (A_a \leftarrow [abs, con^{\cup}]_{\nu}^{\cup}) \quad . \end{array}$$

Hence, by definition,

$$abs_a \bullet \theta(B) \quad \dot{\sqsubseteq} \quad \theta(A) \bullet [abs, con^{\cup}]_{\nu}$$

□

Example – uncurried fold.

Theorem 58 (Fixed Point Fusion) Suppose that, for each type variable a, $(A_a, \sqsubseteq_a)$ is a poset. Let μ denote the polymorphic function that maps (monotonic) function f of type $(A_t, \sqsubseteq_t) \leftarrow (A_t, \sqsubseteq_t)$ to its least fixed point. Then μ is parametric in all pair algebras. That is, given an assignment of Galois connections (abs_a, con_a) between the posets $(A_a, \sqsubseteq_a)$ and $(B_a, \preceq_a)$ for each type variable a and letting P be the corresponding logical pair algebra, we have, for all type expressions t,

$$\mu \in P_{t \leftarrow (t \leftarrow t)} \quad .$$

Also, let ν denote the polymorphic function that maps (monotonic) function f of type $(A_t, \sqsubseteq_t) \leftarrow (A_t, \sqsubseteq_t)$ to its greatest fixed point. Then ν is parametric in all pair algebras. That is, for all type expressions t,

$$\nu \in P_{t \leftarrow (t \leftarrow t)} \quad .$$

Proof

$$\begin{array}{ll}
 & \mu \in P_{t \leftarrow (t \leftarrow t)} \\
= & \quad \{ \quad \text{definition: (1)} \quad \} \\
 & \langle \forall\, f,g \;::\; (\mu f\,,\ \mu g) \in P_t \ \Leftarrow \ (f,\, g) \in P_{t \leftarrow t} \rangle \\
= & \quad \{ \quad \text{lemma 56} \quad \} \\
 & \langle \forall\, f,g \;::\; \\
 & \qquad [abs\,, con^{\cup}]_t \,.\, \mu f \ \sqsubseteq_t \ \mu g \\
 & \qquad \Leftarrow (f,\, g) \in [\![abs^{\cup} \bullet \sqsubseteq\,,\ [abs\,, con^{\cup}]^{\cup}]\!]_{t \leftarrow t} \\
 & \qquad \rangle \\
= & \quad \{ \quad \text{fixed point fusion (see below for explanation)} \quad \} \\
 & \quad true \quad .
\end{array}$$

To see that the last step of the above proof is an instance of the fixed point fusion theorem,

$$h.\mu f \leq \mu g \ \Leftarrow \ h \bullet f \mathrel{\dot{\sqsubseteq}} g \bullet h$$

whenever h is a lower adjoint in a Galois connection, we need to rephrase the premise in the above implication:

$$\begin{array}{ll}
 & (f,\, g) \ \in \ [\![abs^{\cup} \bullet \sqsubseteq\,,\ [abs\,, con^{\cup}]^{\cup}]\!]_{t \leftarrow t} \\
= & \quad \{ \quad \text{definition (47), lemma 56 and theorem 55} \quad \} \\
 & (f,\, g) \ \in \ [abs\,, con^{\cup}]^{\cup}_t \bullet \sqsubseteq_t \leftarrow [abs\,, con^{\cup}]^{\cup}_t \\
= & \quad \{ \quad \text{distributivity: (23)} \quad \} \\
 & (f,\, g) \ \in \ ([abs\,, con^{\cup}]^{\cup}_t \leftarrow A_t) \bullet (\sqsubseteq_t \leftarrow A_t) \bullet (B_t \leftarrow [abs\,, con^{\cup}]^{\cup}_t) \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & ([abs\,, con^{\cup}]_t \bullet f) \ \sqsubseteq_t \leftarrow A_t \ \ (g \bullet [abs\,, con^{\cup}]_t) \\
= & \quad \{ \quad \text{definition (47) and lemma 50} \quad \} \\
 & ([abs\,, con^{\cup}]^{\cup}_t \bullet f) \ [\![\sqsubseteq\,, A]\!]_{t \leftarrow t} \ \ (g \bullet [abs\,, con^{\cup}]_t) \quad .
\end{array}$$

The second claim is dual to the first.
□