Constructive Lattice Theory

Roland Backhouse
Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB Eindhoven,
The Netherlands.
rolandb@win.tue.nl

October 15, 1993

Abstract

A notion of simulation of one datatype by another is defined as a *constructive* preorder. A calculus of datatype simulation is then developed by formulating constructive versions of least-fixed-point theorems in lattice theory. The calculus is applied to the construction of several isomorphisms between classes of datatypes. In particular constructive adaptations of theorems in lattice theory about closure operators are shown to yield simulations and isomorphisms between monad structures, and constructive adaptations of theorems in regular algebra are shown to yield isomorphisms between list structures.

A question to which any respectable theory of datatypes should provide immediate answers is when two datatypes are *isomorphic*, i.e. entirely equivalent modulo implementation details. A subsidiary question is when one datatype *simulates* another. This second question is of interest in its own right but is also important to answering the first question since isomorphism is frequently reduced to mutual simulation.

This paper formulates a number of algebraic laws for the construction of simulations and isomorphisms between datatypes and applies these laws to the construction of several isomorphisms. Among the isomorphisms we construct are the "monad simulation theorem" and the "monad decomposition theorem". These are general theorems that are used to construct isomorphisms between monads. More specific isomorphisms we construct are "list decomposition" and "list leapfrog". "List decomposition" expresses an elementary isomorphism between list structures an instance of which is the solution to the so-called "lines-unlines problem" [13]. "List leapfrog" has a similar elementary interpretation. Surprisingly, I have been unable to find any mention of the monad decomposition theorem in the literature (either in articles or texts on category theory or on lattice theory); even if the theorem is "well-known" its central importance does not seem to be properly recognised.

The theorems we present were invented by the following process: we studied theorems in lattice theory about least fixed points and closure operators with a view to whether the theorems could be adapted to "constructive" theorems about "map relators" and "monads" in our theory. Confidence that this might prove to be a fruitful process came from the idea that category theory equals constructive lattice theory.

This idea is not new. In 1968, Lambek [19] began an article with the sentence

> Classical results on lattices provide a fruitful source of inspiration for discoveries about categories, in view of the fact that a partially ordered set may be regarded as a category in which there is at most one map between any two objects.

In Scott's work [27] there is a clear progression from lattice theory to category theory, and Smyth and Plotkin [28] acknowledge "the well-known analogy between partial orders and categories" as the basis of their generalisation of the solution of fixed point equations to the construction of initial fixed points. In the textbook by Rydeheard and Burstall [26], on the other hand, the idea that category theory is constructive is the explicit theme although they do not make the link with lattice theory. Most recently, Pratt [23] has observed the relationship between the Curry-Howard isomorphism between propositions and types, the well-known focal point of constructive type theory, and residuated lattices.

In outline, we view datatypes (specifically "relators" in our calculus) as "constructively" monotonic functions. The simulation relation is a constructive preorder on datatypes, i.e. it is "constructively" reflexive, "constructively" transitive, and "constructively" monotonic with respect to function composition. By "constructive" we mean that in order to show that one datatype simulates another it is necessary to exhibit a "witness" to the simulation, i.e. a program that allows computations defined on the one datatype to be replaced by computations on the other. Note, however, that the simulation relation is not anti-symmetric because that two datatypes simulate each other does not mean that they are isomorphic; only when the two witnesses are inverse is that the case. The witnesses to simulations are natural transformations, in the sense of category theory.

In this paper we calculate a significant number of non-trivial isomorphisms between (classes of) datatypes in order to substantiate these ideas. The original inspiration for the examples we discuss came from regular algebra [14, 18], the algebra of regular languages. The principal idea is that, by analogy with the Curry-Howard isomorphism between propositions and types, there is an "isomorphism" between regular algebra and the theory of lists. For example, it should be possible to augment a regular-algebra proof of the identity

$$x^* \times x \;=\; x \times x^*$$

with the construction of a bijection that maps a pair consisting of a list and a single value (of the same type as the elements of the list) into a pair consisting of a single value and a list, whereby — in the case of a non-empty list — the returned value is the first element of the given list, and the returned list is the given list with its first element removed and the given value appended at the end of the list. (Thus the pair $([a,b,c],\ d)$, for example, should be mapped to the pair $(a,\ [b,c,d])$.)

The construction of programs on lists that witness identities of regular algebra is not a difficult task. The challenge is to develop the theory that clearly links the activities of proof in regular algebra and the construction of programs on lists. A first step to meet this challenge was to develop a novel theory of datatypes purposefully designed to expedite program calculation.

The use of the so-called "spec-calculus", which forms the basis for our work, is demonstrated in the calculations that follow. By design, programs developed within the spec calculus are typically more compact than programs written in conventional programming languages.

Moreover, the emphasis in the spec calculus is on the recognition of concepts and programming constructs that are relevant to a broad class of problems involving non-trivial datatypes. The consequence is that programs derived in the spec calculus cannot always be directly translated into programs in any existing programming language. Fortunately it is always possible to transform the spec calculus expressions we derive into equal expressions that are implementable but, unfortunately, this process sometimes involves long and arduous calculations. It is not the purpose of this paper to go into these problems in depth but we do discuss the issue in the section on monad decomposition if only to convince the reader that we are constructing "real" programs.

The current paper is not entirely self-contained. A summary of the main elements of the spec calculus is given in section 1 which should suffice for those not wishing to check every detail of the proofs we present. Otherwise reference may be made to the various articles that have been published to date about the calculus [6, 5, 10, 9, 16, 17, 1, 29, 30] of which [1] is the most comprehensive. We also exploit the theory of reductivity developed in [8].

The paper begins properly in section 2 by motivating definitions of the notions of simulation and isomorphism as *constructive* preorders, and presenting the specific definitions we use in the spec calculus. This is followed by a brief summary of fixed point theorems in lattice theory. (For the most part these are discussed in detail in part 1 of [1].) Thereafter, all of these theorems are made constructive, following which we present the applications. Before reading the constructive calculations the reader would be well-advised to take the stated theorems and develop a non-constructive proof using the lattice-theoretic laws. This is how the calculations were initially designed and it helps considerably to understand the structure of the proofs.

1 The Spec Calculus and Relators

The spec calculus is a point-free (and thus incomplete) axiomatisation of the calculus of relations augmented with axioms asserting the existence of a unit type, $\mathbb{1}$, a disjoint sum operator, $+$, and a (non-categorical) cartesian product operator, $\times$. In other words, the spec calculus is an algebra with model the class of binary relations over some universe U where U is non-empty and closed under two "tagging" operators *inl* ("inject left") and *inr* ("inject right") and under pair formation.

Specific details of the calculus that are essential to the current discussion are as follows. (Lots of inessential —to the current discussion— details are omitted and this should not be seen as a complete summary of the calculus.) The carrier set of the calculus forms a complete universally distributive lattice ordered by the relation $\sqsubseteq$. Objects of the carrier set are called *specs* (which is an abbreviation of specifications). The supremum operator of the lattice is denoted by $\sqcup$ and its infimum operator by $\sqcap$. The top element of the lattice is denoted by $\top\!\!\top$ and its bottom element by $\perp\!\!\perp$. (We prefer these symbols to the more conventional $\top$ and $\perp$ because in hand-written documents the former is easily confused with the letter T.) The carrier set also includes a distinguished element I (different from $\perp\!\!\perp$ and $\top\!\!\top$). The interpretations of $\top\!\!\top$, $\perp\!\!\perp$ and I in the relational model are the universal relation (the relation holding between all pairs), the empty relation and the identity relation, respectively.

Specs are closed under the unary operator $^{\cup}$ (written as a postfix to its argument) and the binary operator $\circ$. The interpretation[1] of $\cup$ is the relational converse operator and the interpretation of $\circ$ is relational composition. The set of specs forms a monoid under $\circ$ with I as the unit element. Composition distributes universally through the supremum operator. Moreover, $\cup$ is a self-inverse lattice isomorphism that respects I and distributes contravariantly through $\circ$. The operators $\cup$, $\sqcap$ and $\circ$ are related to each other by the rule called "Dedekind's rule" by Riguet [25] and "the modular identity" by Freyd and Scedrov [15].

To axiomatise disjoint sum two specs $\hookrightarrow$ and $\hookleftarrow$ are postulated. These have interpretations the two polymorphic tagging functions inject left and inject right, respectively. The binary operators $\triangledown$, $\blacktriangledown$ and $+$ are then defined by the equations

(1) $\quad R \triangledown S \;=\; (R \circ \hookrightarrow\cup) \sqcup (S \circ \hookleftarrow\cup) \quad ,$

(2) $\quad R \blacktriangledown S \;=\; (\hookrightarrow \circ R) \sqcup (\hookleftarrow \circ S) \quad$, and

(3) $\quad R + S \;=\; (\hookrightarrow \circ R \circ \hookrightarrow\cup) \sqcup (\hookleftarrow \circ S \circ \hookleftarrow\cup) \quad .$

Their properties are axiomatised by the laws

(4) $\quad (\hookrightarrow \circ \hookrightarrow\cup) \;\sqcup\; (\hookleftarrow \circ \hookleftarrow\cup) \;\sqsubseteq\; I \quad$, and

(5) $\quad (R \triangledown S) \circ (T \blacktriangledown U) \;=\; (R \circ T) \sqcup (S \circ U) \quad .$

In category-theory texts $R \triangledown S$ would be written $[R, S]$. There is however no standard notation that we are aware of for $R \blacktriangledown S$.

Dual to disjoint sum, cartesian product is axiomatised by postulating the existence of two specs $\ll$ and $\gg$. These have interpretations the two projection functions mapping a pair to its left component and right component, respectively. Then the three binary operators $\vartriangle$, $\blacktriangle$ and $\times$ are defined by the equations

(6) $\quad R \vartriangle S \;=\; (\ll\cup \circ R) \sqcap (\gg\cup \circ S) \quad ,$

(7) $\quad R \blacktriangle S \;=\; (R \circ \ll) \sqcap (S \circ \gg) \quad$, and

(8) $\quad R \times S \;=\; (\ll\cup \circ R \circ \ll) \sqcap (\gg\cup \circ S \circ \gg) \quad .$

Finally, their properties are axiomatised by the laws

(9) $\quad (\ll\cup \circ \ll) \;\sqcap\; (\gg\cup \circ \gg) \;\sqsubseteq\; I$

(10) $\quad (R \blacktriangle S) \circ (T \vartriangle U) \;=\; (R \circ T) \sqcap (S \circ U) \quad$, and

(11) $\quad \top\!\!\top \circ \ll \;=\; \top\!\!\top \circ \gg \quad .$

Again, there is a standard notation in category-theory texts for the analogous operation to $R \vartriangle S$, viz. $\langle R, S\rangle$, but so far as we know there is no standard notation for $R \blacktriangle S$.

A (unary) *relator*, F, is by definition a function from specs to specs satisfying the four properties

(12) $\quad F.I \;\sqsubseteq\; I \quad ,$

[1] From here on references to "the interpretation" will mean "the interpretation in the relational model"

(13) $\quad F.(R \circ S) \;=\; F.R \circ F.S$,

(14) $\quad F.R \sqsubseteq F.S \;\Leftarrow\; R \sqsubseteq S$, and

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

Relators of higher arity (binary, ternary etc.) obey similar laws obtained by replacing all constants and variables by vectors of the appropriate arity and lifting all operators elementwise to operate on vectors.

A spec A is called a *monotype* if $A \sqsubseteq I$. (Freyd and Scedrov use the term *coreflexive*.) A spec f is called an *imp* if $f \circ f^\cup \sqsubseteq I$, a *co-imp* if $f^\cup$ is an imp and a *bijection* if it is both an imp and a co-imp. (Freyd and Scedrov use the term *simple* for a co-imp and *isomorphism* for a bijection.) A frequently used property is that $A \circ A \;=\; A$ for all monotypes A.

It is easy to show that relators preserve monotypes, imps, co-imps and bijections. Monotypes may be interpreted as sets, imps and co-imps as (partial) functions and "co-"functions. (Whether you interpret imps as functions or co-imps as functions depends on whether you choose to view the "input" to a relation as being the left or right component of a pair.)

If A is a monotype then the constant function $(X \mapsto A)$ is a relator. Disjoint sum $(+)$ and cartesian product $(\times)$ are also relators. The identity function is trivially a relator. Fixing one argument of an $(n+1)$-ary relator to a monotype yields an n-ary relator, and functional composition of two relators yields a relator. Finally the least-fixed point, μF, of a unary relator, F, is a monotype, and if $\oplus$ is an $(n+1)$-relator then the function $(R \mapsto \mu(S \mapsto R \oplus S))$ is an n-ary relator (where R ranges over n-ary vectors of specs and S over specs). (Fixed points always exist because the specs are assumed to form a complete lattice.)

Two derived operators play a fundamental rôle in the calculus: the *left* and *right domain* operators, denoted by the postfix symbols $<$ and $>$, respectively. They have the characterising properties:

(16) $\quad \forall(A \,:\, A \sqsubseteq I \,:\, A \circ R = R \;\equiv\; A \sqsupseteq R<)$, and

(17) $\quad \forall(A \,:\, A \sqsubseteq I \,:\, R \circ A = R \;\equiv\; A \sqsupseteq R>)$.

Our left domain operator corresponds to Freyd and Scedrov's *Dom* operator. The monotypes $R<$ and $R>$ should not be confused with their $\Box R$ and $R\Box$. An important property of the domain operators is that they commute with relators. I.e. for all relators F and all specs R

(18) $\quad F.(R<) \;=\; (F.R)<$, and $\quad F.(R>) \;=\; (F.R)>$

(These are not axioms but derived properties.)

Two programming constructs that also play a fundamental rôle in the calculus are the so-called *catamorphisms* and *anamorphisms*. For given endorelator F and spec R, $(\!|F;\; R|\!)$ is defined to be the least solution of the equation $X :: \quad X \;=\; R \circ F.X$ and $[\![F;\; R]\!]$ is defined to be the least solution of the equation $X :: \quad X \;=\; F.X \circ R$. The function $R \mapsto (\!|F;\; R|\!)$ is thus a total function from specs to specs. Its range is the set of F-catamorphisms. The function $R \mapsto [\![F;\; R]\!]$ is also a total function from specs to specs, its range being the set of F-anamorphisms. The relation between the two is that $[\![F;\; R^\cup]\!] \;=\; (\!|F;\; R|\!)^\cup$. For a variety of programming applications of catamorphisms and anamorphisms —in the setting of a calculus of total functions— the reader is referred to [21].

A brief summary of our notational conventions is necessary. A general rule we use is that prefix and postfix operators have the highest operator precedence. Function application is denoted by an infix dot, e.g. $f.x$, function composition by a raised infix dot, e.g. $f \cdot g$. The metalanguage we use is the predicate calculus, and here we use standard notational conventions. Operators of the predicate calculus all have lower precedence than operators of the spec calculus. The composition operator in the spec calculus has lower precedence than that of the binary relators, all of which have the same precedence. (For example, $R \circ S{\oplus}T$ should be parsed as $R \circ (S{\oplus}T)$.) Some effort has been put into the spacing of terms in formulae so that what the eye sees is the intended syntactic structure.

For brevity we often use square brackets to denote abstraction with respect to spec-valued variables in a formula. For example we write $[*R \times R]$ to denote the relator $(R \mapsto *R \times R)$. Where the enclosed formula contains more than one free variable the square bracket notation denotes abstraction in alphabetic order. Thus $[R{\times}S]$ denotes the binary relator $(R \mapsto (S \mapsto R{\times}S))$ and is different to $[S{\times}R]$ which denotes the relator $(R \mapsto (S \mapsto S{\times}R))$. This notation has been introduced for brevity and to make the calculations with functions (relators) take on the outward appearance of calculations with elements in the regular algebra.

2 Simulation = Constructive Preorder

Lattice theory is a theory about preorders. We want to view simulations as *constructive* preorders on datatypes obeying *coherent* composition rules. That is, if F and G are datatypes then F "simulates" G if there is some program γ, henceforth called the *witness*, that enables one to transform all computations involving G-structures to computations involving F-structures. Furthermore, the simulates relation should be "constructively" reflexive, transitive and monotonic.

"Constructively" reflexive just means that every datatype must simulate itself. "Constructively" transitive means that if datatype F simulates datatype G, and datatype G simulates datatype H, then F should simulate H, the witness being formed by some composition of the two given witnesses. "Constructively" monotonic means that if datatype F simulates datatype G, and datatype H simulates datatype K, then $F \cdot H$ should simulate $G \cdot K$ (where $\cdot$ denotes composition of functions). Again the witness must be formed by some composition of the two given witnesses. There are thus two ways of composing witnesses which — borrowing the terminology from category theory — we will call *horizontal* and *vertical* composition of the witnesses.

The final condition that we demand of simulations is that different ways of composing them should be *coherent*. This boils down to four requirements: the witness to reflexivity should be a unit of horizontal composition, both horizontal and vertical composition should be associative, and horizontal and vertical composition of witnesses should satisfy a so-called interchange law. (The terms "interchange law" and "coherent" are also borrowed from category theory. See [20].)

To explain the second of these requirements suppose we have three simulations $F_0 \gtrsim F_1$, $F_1 \gtrsim F_2$ and $F_2 \gtrsim F_3$. Then, by transitivity, we have $F_0 \gtrsim F_3$. However, there are two ways to arrive at this conclusion. We can first compose the first two simulations and then compose the result ($F_0 \gtrsim F_2$) with the third, or we can first compose the last two simulations and compose the result $(F_1 \gtrsim F_3)$ with the first simulation. Associativity of horizontal composition guarantees that the order is irrelevant.

A similar argument holds for vertical composition. Suppose we have three simulations $F_0 \gtrsim G_0$, $F_1 \gtrsim G_1$ and $F_2 \gtrsim G_2$. Then, using monotonicity, there are two ways to arrive at the conclusion $F_0 \cdot F_1 \cdot F_2 \gtrsim G_0 \cdot G_1 \cdot G_2$. One involves arguing that $(F_0 \cdot F_1) \cdot F_2 \gtrsim (G_0 \cdot G_1) \cdot G_2$, the other that $F_0 \cdot (F_1 \cdot F_2) \gtrsim G_0 \cdot (G_1 \cdot G_2)$. (Note the parenthesisation.) Again, the coherence requirement that vertical composition of witnesses is associative guarantees that the two witnesses are equal. An interchange law between horizontal and vertical composition is demanded by the requirement that the composition of witnesses to the four simulations $F_0 \gtrsim F_1$, $F_1 \gtrsim F_2$, $G_0 \gtrsim G_1$ and $G_1 \gtrsim G_2$ to form a witness to the simulation $F_0 \cdot G_0 \gtrsim F_2 \cdot G_2$ should be independent of the order of composition.

The coherence requirements do not imply unicity of witnesses. They do however guarantee that any design choice in the construction of witnesses is intrinsic to the particular datatypes involved and not dependent on the order of application of the basic composition primitives.

Adopting this view of a theory of datatypes means that the theory subsumes lattice theory, the latter being a particular case in which witnesses are unique (and of no interest). More importantly, it means that lattice theory is, in Lambek's words, a "source of inspiration for discoveries about" simulations between datatypes.

There are several possibilities for the notion of simulation in the spec calculus. For concreteness we assume the following definition throughout this paper.

Definition 19 (Natural Simulation) Let F and G be relators and let γ be a spec. We say that γ witnesses the fact that F simulates G and write $\gamma \in F \gtrsim G$ iff

(a) $\gamma \in F \mathrel{\dot{\leftrightarrow}} G$,

(i.e. $\forall(R :: F.R \circ \gamma \;=\; \gamma \circ G.R)$) , and

(b) $\gamma{>} \;=\; G.I$.

□

Note that is immediate from (a) and (b) that $\gamma{<} \sqsubseteq F.I$ and, hence, $F.I \circ \gamma \;=\; \gamma$. Note also that we do not stipulate that γ is an imp or a co-imp. (In most cases it is very easy to determine whether a spec is an imp or a co-imp, and adding such a clause to the definition has little value.)

Definition 20 (Natural Isomorphism) Let F and G be relators and let γ be a spec. We say that γ witnesses the fact that F is isomorphic to G and write $\gamma \in F \cong G$ iff

(a) $\gamma \in F \gtrsim G$

(b) $\gamma{<} \;=\; F.I$, and

(c) γ is a bijection.

$\square$

We shall say that F *is isomorphic to* G if a γ exists such that $\gamma \in F \cong G$.

Note that the definition of isomorphism we use implies but is not equivalent to mutual simulation – simulation is not required to be "constructively" anti-symmetric.

Theorem 21 The relations $\dot{\leftrightarrow}$, $\gtrsim$ and $\cong$ are all constructive preorderings on relators. Specifically, each is constructively reflexive: for all $\trianglerighteq \in \{\dot{\leftrightarrow}, \gtrsim, \cong\}$ and all relators F ,

$$F.I \in F \trianglerighteq F \quad .$$

Each is constructively transitive: for all $\trianglerighteq \in \{\dot{\leftrightarrow}, \gtrsim, \cong\}$, all specs γ and δ, and all relators F, G, and H,

$$\gamma \in F \trianglerighteq G \;\wedge\; \delta \in G \trianglerighteq H \quad \Rightarrow \quad \gamma \circ \delta \in F \trianglerighteq H \quad .$$

Finally, composition of relators is constructively monotonic in both its arguments with respect to each of the three relations: for all $\trianglerighteq \in \{\dot{\leftrightarrow}, \gtrsim, \cong\}$, all specs γ and δ, and all relators F_0, G_0, F_1 and G_1,

$$\gamma \in F_0 \trianglerighteq G_0 \;\wedge\; \delta \in F_1 \trianglerighteq G_1 \quad \Rightarrow \quad \gamma \circ G_0.\delta \;\in\; F_0 \cdot F_1 \;\trianglerighteq\; G_0 \cdot G_1$$

$\square$

The proof of this theorem is straightforward and hence omitted. In proof hints the adjective "constructively" will sometimes be omitted.

The introduction of the relation $\dot{\leftrightarrow}$ on relators allows a useful decomposition of proof obligations when reasoning about simulations and isomorphisms. Nevertheless, the relation is not by itself important because, in spite of it being a constructive preorder, composition of witnesses to this relation is not in general coherent. Composition of simulations and isomorphisms is. As an illustration of the sort of argument that is needed to verify this claim suppose we are given three simulations/isomorphisms: $\alpha \in F_0 \trianglerighteq F_1$, $\beta \in F_1 \trianglerighteq F_2$, $\gamma \in G \trianglerighteq H$. Then we can combine these in two different ways to form a witness to $F_0 \cdot G \trianglerighteq F_2 \cdot H$. First,

$$
\begin{array}{ll}
 & \alpha \in F_0 \trianglerighteq F_1 \quad\wedge\quad \beta \in F_1 \trianglerighteq F_2 \quad\wedge\quad \gamma \in G \trianglerighteq H \\
\Rightarrow & \qquad \{ \qquad \text{transitivity} \quad \} \\
 & \alpha \circ \beta \in F_0 \trianglerighteq F_2 \quad\wedge\quad \gamma \in G \trianglerighteq H \\
\Rightarrow & \qquad \{ \qquad \text{monotonicity} \quad \} \\
 & (\alpha \circ \beta) \circ F_2.\gamma \;\in\; F_0 \cdot G \;\trianglerighteq\; F_2 \cdot H \quad .
\end{array}
$$

Second,

$$
\begin{array}{ll}
 & \alpha \in F_0 \trianglerighteq F_1 \quad\wedge\quad \beta \in F_1 \trianglerighteq F_2 \quad\wedge\quad \gamma \in G \trianglerighteq H \\
\Rightarrow & \qquad \{ \qquad \text{monotonicity and reflexivity} \quad \} \\
 & \quad \alpha \circ F_1.G.I \;\in\; F_0 \cdot G \;\trianglerighteq\; F_1 \cdot G
\end{array}
$$

$$
\begin{array}{ll}
& \wedge\ \beta \circ F_2.G.I \ \in\ F_1 \cdot G \ \trianglerighteq\ F_2 \cdot G \quad \wedge \quad \gamma \in G \trianglerighteq H \\
\Rightarrow & \qquad \{ \qquad \text{transitivity and monotonicity} \quad \} \\
& \quad (\alpha \circ F_1.G.I) \circ (\beta \circ F_2.G.I) \ \in\ F_0 \cdot G \ \trianglerighteq\ F_2 \cdot G \\
& \wedge\ F_2.I \circ F_2.\gamma \ \in\ F_2 \cdot G \ \trianglerighteq\ F_2 \cdot H \\
\Rightarrow & \qquad \{ \qquad \text{transitivity} \quad \} \\
& ((\alpha \circ F_1.G.I) \circ (\beta \circ F_2.G.I)) \circ F_2.I \circ F_2.\gamma \ \in\ F_0 \cdot G \ \trianglerighteq\ F_2 \cdot H \quad .
\end{array}
$$

The two witnesses are however equal as verified by the following simple calculation.

$$
\begin{array}{ll}
& ((\alpha \circ F_1.G.I) \circ (\beta \circ F_2.G.I)) \circ F_2.I \circ F_2.\gamma \\
= & \qquad \{ \qquad \text{associativity of composition} \quad \} \\
& \alpha \circ (F_1.G.I \circ \beta) \circ F_2.G.I \circ F_2.I \circ F_2.\gamma \\
= & \qquad \{ \qquad \bullet \quad \beta \in F_1 \trianglerighteq F_2 \text{ , associativity of composition} \quad \} \\
& \alpha \circ \beta \circ (F_2.G.I \circ F_2.G.I \circ F_2.I) \circ F_2.\gamma \\
= & \qquad \{ \qquad \text{properties of monotypes and relators} \quad \} \\
& \alpha \circ \beta \circ F_2.G.I \circ F_2.\gamma \\
= & \qquad \{ \qquad F_2 \text{ is a relator, } \bullet \quad \gamma \in G \trianglerighteq H \text{ , thus } G.I \circ \gamma \ =\ \gamma \quad \} \\
& \alpha \circ \beta \circ F_2.\gamma \quad .
\end{array}
$$

The definitions of "simulates" and "is isomorphic to" are stated in the form appropriate to *unary* relators but, following the usual practice, we intend the definitions to apply also to relators of higher arity. In fact, the definitions need only be given for unary relators because for higher order relators we can reduce the notions to a conjunction of conditions each of which is stated in terms of unary relators. We illustrate this with the definition of isomorphism between two binary relators.

Definition 22 Let $\oplus$ and $\otimes$ be binary relators, and let γ be a spec. Then we say that γ witnesses the fact that $\oplus$ is isomorphic to $\otimes$ and write $\gamma \in \oplus \cong \otimes$ iff

(a) $\gamma \ \in\ I\oplus \ \cong\ I\otimes$, and

(b) $\gamma \ \in\ \oplus I \ \cong\ \otimes I$.

□

(In general for a pair of n-ary relators take the n pairs of unary relators formed by fixing all but one argument to I. The witness must simultaneously witness an isomorphism between the n pairs.)

It is easy to verify that $\gamma \in \oplus \cong \otimes$ is equivalent to the three conditions:

(a) $\forall(R,S ::\ R{\oplus}S \circ \gamma \ =\ \gamma \circ R{\otimes}S)$

(b) $\gamma{<} \ =\ I{\oplus}I$, and

(c) $\gamma{>} \ =\ I{\otimes}I$.

We conclude this section with a useful property of disjoint sum. In fact, the property is an instance of a "constructive Galois connection", but, in order to focus our discussion on constructive fixed point theorems, we shall not amplify here on this remark.

Theorem 23 Let α, β and γ be specs, and let F, G and H be relators. Then,

(a) $\gamma \circ \hookrightarrow \;\in\; F \gtrsim G \;\;\wedge\;\; \gamma \circ \hookleftarrow \;\in\; F \gtrsim H \quad\Leftarrow\quad \gamma \;\in\; F \gtrsim G{+}H$,

(b) $\alpha \triangledown \beta \;\in\; F \gtrsim G{+}H \quad\Leftarrow\quad \alpha \in F \gtrsim G \;\;\wedge\;\; \beta \in F \gtrsim H$.

□

3 Fixed Point Theorems

This section contains a short review of lattice-theoretic fixed-point theorems that we intend to adapt to "constructive" fixed point theorems. It is helpful if the reader is fully conversant with the use of these theorems since their constructive forms are more complex and thus less easy to understand and to use. Nevertheless, since the topic of the paper is indeed the use of the constructive theorems, the summary is very brief. The section also includes a list of properties of catamorphisms and anamorphisms that are corollaries of the fixed point theorems, but stated without proof. The reader not already familiar with the theorems may find it instructive to undertake the exercise of proving a selection of them.

3.1 A Small Calculus

The first, and undoubtedly most important, fixed point theorem is that due to Knaster and Tarski. The theorem states that if $f \in \mathcal{A} \leftarrow \mathcal{A}$ is a monotonic endofunction on complete lattice $(\mathcal{A}, \geq)$ then f has a least fixed point, denoted μf , characterised by the properties

(24) $\mu f \;=\; f.\mu f$, and

(25) $x \geq \mu f \;\Leftarrow\; x \geq f.x$ for all $x \in \mathcal{A}$.

Starting from the Knaster-Tarski theorem one can develop quite a rich fixed-point calculus. There are three rules of this calculus that will be particularly important to us.

The *rolling rule* states that for all monotonic functions f and g,

(26) $f \,.\, \mu(g \cdot f) \;=\; \mu(f \cdot g)$.

(Of course, the domain of f must equal the range of g , and vice-versa, and both domains must form complete lattices. Such details will be omitted in this summary.)

The *fusion law* states that, for all monotonic functions f and h, and all universally $\sqcup$-junctive functions g (thus functions g having an upper Galois adjoint),

(27) $\mu f \;=\; g.\mu h \quad\Leftarrow\quad f \cdot g \;=\; g \cdot h$.

Finally, the *diagonal rule* states that, for all monotonic, binary operators $\oplus$,

(28) $\mu(x \mapsto x \oplus x) \;=\; \mu(x \mapsto \mu(y \mapsto x \oplus y))$.

As is well known, existence of fixed points is also guaranteed within the weaker context of a CPO (rather than a complete lattice). The fusion law also remains valid if the requirement on g is weakened to it being strict and continuous. (This seems to be less well known — rather, I must admit to myself having been ignorant of this fact — since in computing texts it is usually required that all three functions f, g and h be continuous whereas monotonicity of f and h is sufficient.) See the discussion on the constructive fusion law for why we do not state the rule in this form. (The fusion law also remains valid when both occurrences of "=" are replaced by " $\geq$", but we have no use in the current paper for this version of the law.)

3.2 Applications to Catamorphisms and Anamorphisms

Exercises in the use of these laws that occur in the forthcoming calculations are the following. Recall that, by definition,

$$(\!|F;\ R|\!) = \mu(X \mapsto R \circ F.X)\ \text{ , and } \qquad [\![F;\ R]\!] = (\!|F;\ R^{\cup}|\!)^{\cup}\ .$$

for all relators F and all specs R. Then an instance of the rolling rule is the *catamorphism rolling rule*:

(29) $\qquad F.(\!|G \cdot F;\ R|\!) = (\!|F \cdot G;\ F.R|\!)\ .$

Instances of the fusion law are manyfold. Examples are: For all relators F and all specs R:

(30) $\qquad [\![F;\ R]\!] \ = \ \mu(X \mapsto F.X \circ R)\ ,$

and for all relators F and G, and all specs R:

(31) $\qquad [\![F;\ R]\!] = (\!|G;\ R|\!) \ \Leftarrow \ R \in F \mathrel{\dot{\leftrightsquigarrow}} G\ .$

(Use the fact that converse is universally $\sqcup$-junctive.)

(32) $\qquad (\!|F;\ R|\!) \circ [\![F;\ S]\!] \ = \ \mu(X \mapsto R \circ F.X \circ S)\ .$

(Use the fact that $(X\circ)$ is universally $\sqcup$-junctive for all X.) For all binary relators $\oplus$:

(33) $\qquad (X \mapsto (\!|I\oplus;\ X\oplus I|\!))$ is a relator.

For all relators F and G, and all specs R and S ,

(34) $\qquad (\!|F;\ R|\!) \circ (\!|G;\ S|\!) \ = \ (\!|G;\ R \circ S|\!) \ \Leftarrow \ S \in F \mathrel{\dot{\leftrightsquigarrow}} G\ .$

For all binary relators $\oplus$, and all specs R and S ,

(35) $\qquad (\!|I\oplus;\ R|\!) \circ (\!|I\oplus;\ S\oplus I|\!) \ = \ (\!|I\oplus;\ R \circ S\oplus I|\!)\ .$

(A corollary of (34).) For all binary relators $\oplus$, all specs R and all monotypes A,

(36) $\qquad (\!|I\oplus;\ R|\!) \circ (\!|I\oplus;\ A\oplus I|\!) \ = \ (\!|A\oplus;\ R|\!)\ .$

Finally, an example of the use of the diagonal rule, in combination with (35), is, for all binary relators $\oplus$, and all specs R,

(37) $\qquad (\!|[\mu(X\oplus)];\ (\!|I\oplus;\ R|\!)|\!) \ = \ (\!|[X\oplus X];\ R|\!)\ .$

(Recall our use of the brackets [and] to denote abstraction with respect to the free variables in a formula.)

3.3 Binary Operators

Suppose $\oplus$ and $\otimes$ are monotonic binary operators and let x be a lattice element. Then the sections $(x\oplus)$ and $(x\otimes)$ are monotonic functions and, of course, the rules stated in subsection 3.1 can be instantiated with these functions replacing the function variables. For example, instantiating f and h in the fusion law (27) by $(x\oplus)$ and $(x\otimes)$, respectively, we obtain: for all universally $\sqcup$-junctive functions g,

$$(38) \qquad \mu(x\oplus) \;=\; g.\mu(x\otimes) \;\Leftarrow\; \forall(y :: \; x \oplus g.y \;=\; g.(x\otimes y)) \;\; .$$

Nothing is lost by this instantiation since (27) can be recovered from (38) as follows: given functions f and h define the binary operators $\oplus$ and $\otimes$ by $x\oplus y = f.x$ and $x\otimes y = h.x$ and then instantiate (38) with these definitions. The two laws (27) and (38) are formally equivalent. It turns out, however, that the constructive formulations of these laws are not equivalent. Indeed full generality — in the constructive case — can only be obtained by going one step further than (38). Specifically, the law that we generalise is obtained from (38) by instantiating the function g with the section $(\oslash x)$ where $\oslash$ is a monotonic binary function. Assuming that $(\oslash x)$ is universally $\sqcup$-junctive we have:

$$(39) \qquad \mu(x\oplus) \;=\; \mu(x\otimes)\oslash x \;\Leftarrow\; \forall(y :: \; x\oplus(y\oslash x) = (x\otimes y)\oslash x) \;\; .$$

The same is true for the Knaster-Tarski theorem. Instead of adapting (25) we adapt the equivalent

$$(40) \qquad x \geq \mu(y\oplus) \;\Leftarrow\; x \geq y\oplus x \;\; .$$

3.4 Closure Operators

Closure operators form a significant class of monotonic endofunctions. Given any monotonic endofunction f on a complete lattice the least closure operator majorising f is denoted by $f^\star$ and is defined by

$$f^\star.x \;=\; \mu(y \mapsto x \sqcup f.y) \;\; ,$$

where $\sqcup$ denotes the supremum operator in the lattice. Among the many properties of closure operators we single out the closure decomposition rule [1] , for all monotonic endofunctions f and g,

$$(f\sqcup g)^\star \;=\; g^\star \cdot (f \cdot g^\star)^\star \;\; ,$$

for special attention.

"Constructive" closure operators are monads (sometimes called triples [12]). In section 5 we show that every relator F defines a monad $F^\star$, and we prove a monad simulation theorem expressing a minimality property of monads. Further, we discuss in detail the monad decomposition theorem and the relationship between spec-calculus witnesses and programs that might be written in a functional programming language to witness the decomposition.

4 Constructive Fixed Point Theorems

We are now ready to present the constructive fixed point theorems. We begin in the next section with the constructive version of (40). Several special cases are then consider among which the constructive form of (25). Subsequently we formulate and prove the constructive form of (39). Several special cases of this rule are also given. Finally, we consider the rolling and diagonal rules. It is unnecessary to formulate constructive versions of these rules in the spec calculus but we explain why such versions are likely to be necessary in other systems.

4.1 The Constructive Knaster-Tarski Theorem

Assume that $\oplus$ is a binary relator. Then the function ϖ defined by $\varpi.S = \mu(X \mapsto S\oplus X)$ is a relator. (See (33).) The constructive Knaster-Tarski theorem gives a condition under which a simulation of ϖ by a relator F can be constructed.

Theorem 41 (Constructive Knaster-Tarski) For all specs γ, relators F and binary relators $\oplus$,

$$(\!|I\oplus;\ \gamma|\!) \in F \stackrel{>}{\sim} [\mu(S \mapsto R\oplus S)] \quad \Leftarrow \quad \gamma \in F \stackrel{>}{\sim} [R \oplus F.R] \ .$$

□

Note that $[\mu(S \mapsto R\oplus S)]$ denotes the unary relator $(R \mapsto \mu(S \mapsto R\oplus S))$. (See the introduction for discussion of this notational convention.) The relator $[R \oplus F.R]$ will occasionally be denoted by $\mathbb{I} \oplus F$, $\mathbb{I}$ denoting the identity relator.

To prove the theorem we have to check two conditions: the naturality of $(\!|I\oplus;\ \gamma|\!)$ and its totality over $\mu(I\oplus)$. The naturality property is a property that we use independently of theorem 41 and so is stated in the following lemma.

Lemma 42 For all specs γ, R and S , and all binary relators $\oplus$,

$$(\!|I\oplus;\ \gamma|\!) \in R \leftrightsquigarrow \mu(X \mapsto S\oplus X) \quad \Leftarrow \quad \gamma \in R \leftrightsquigarrow S\oplus R \ .$$

Proof

$$\begin{array}{ll}
 & (\!|I\oplus;\ \gamma|\!) \in R \leftrightsquigarrow \mu(X \mapsto S\oplus X) \\
\equiv & \{ \quad \text{definition of } \leftrightsquigarrow \quad \} \\
 & R \circ (\!|I\oplus;\ \gamma|\!) = (\!|I\oplus;\ \gamma|\!) \circ \mu(X \mapsto S\oplus X) \\
\equiv & \{ \quad \text{map fusion: (35)}, \mu(X \mapsto S\oplus X) = (\!|I\oplus;\ S\oplus I|\!) \quad \} \\
 & R \circ (\!|I\oplus;\ \gamma|\!) = (\!|I\oplus;\ \gamma \circ S\oplus I|\!) \\
\Leftarrow & \{ \quad \text{catamorphism fusion: [1]} \quad \} \\
 & R \circ \gamma = \gamma \circ S\oplus I \circ I\oplus R \\
\equiv & \{ \quad \oplus \text{ is a relator} \quad \} \\
 & R \circ \gamma = \gamma \circ S\oplus R \\
\equiv & \{ \quad \text{definition of } \leftrightsquigarrow \quad \} \\
 & \gamma \in R \leftrightsquigarrow S\oplus R \ .
\end{array}$$

$\Box$

Note that lemma 42 is itself a constructive Knaster-Tarski theorem in that $\Leftrightarrow$ is a constructive preorder: compare the lemma with (40) omitting the witnesses $([I\oplus;\ \gamma])$ and γ.

Lemma 43 For all specs γ,

$$([I\oplus;\ \gamma])\!> \,=\, \mu(I\oplus) \ \Leftarrow\ \exists(F ::\ \gamma\ \in\ F \overset{>}{\sim} I\!I \oplus F)\ \ .$$

Proof

$$
\begin{array}{ll}
& ([I\oplus;\ \gamma])\!> \,=\, \mu(I\oplus) \\
\Leftarrow & \{\quad \text{totality of catamorphisms: [1]} \quad\} \\
& \gamma\!> \ \sqsupseteq\ I \oplus \gamma\!< \\
\Leftarrow & \{\quad \text{calculus} \quad\} \\
& \exists(F ::\ \gamma\!> \ =\ I \oplus F.I \ \ \wedge\ \ F.I\ \sqsupseteq\ \gamma\!<) \\
\Leftarrow & \{\quad \text{definition} \quad\} \\
& \exists(F ::\ \gamma\ \in\ F \overset{>}{\sim} I\!I \oplus F)\ \ .
\end{array}
$$

$\Box$

Combining the two lemmas with the definition of simulation we obtain the constructive Knaster-Tarski theorem:

$$
\begin{array}{ll}
& ([I\oplus;\ \gamma]) \in F \overset{>}{\sim} [\mu(S \mapsto R\oplus S)] \\
\equiv & \{\quad \text{definition of simulation} \quad\} \\
& \quad \forall(R ::\ ([I\oplus;\ \gamma])\ \in\ F.R \Leftrightarrow \mu(S \mapsto R\oplus S)) \\
& \wedge\ ([I\oplus;\ \gamma])\!> \,=\, \mu(S \mapsto I\oplus S) \\
\Leftarrow & \{\quad \text{lemmas 42 and 43} \quad\} \\
& F \overset{>}{\sim} [R \oplus F.R]\ \ .
\end{array}
$$

The constructive Knaster-Tarski theorem, when applied to functions rather than arbitrary specs, is an instance of Reynolds' abstraction theorem [24]. Fixed point constructions in a general categorical framework have been considered by Adámek and Koubek [2] and Schmidt and Plotkin [28]. The added information predicted by Reynolds' theorem is the naturality of the construction (called a theorem "for free" by Wadler [31]). The fact that the definition of catamorphism can be extended to a relational framework, and that the theorem remains valid within such a framework, was presented by the author at a STOP workshop in 1989 and was the main inspiration for the development of the spec calculus. We return to this matter in the concluding section.

4.2 Constructive Fusion Law

Now we establish the constructive form of (39). First, a lemma that we deem to call a theorem because we expect it to be more widely applicable in the future than just to the theorem that follows it.

Theorem 44 For all relators F and all specs γ ,

$$
\begin{array}{ll}
 & (\!|I\oplus;\ \gamma|\!) \in F \cong [\mu(R\oplus)] \\
\Leftarrow & \gamma\ \in\ F \cong I\!\!I\oplus F \quad\wedge\quad \gamma^{\cup} \text{ is } (I\oplus)\text{-reductive.}
\end{array}
$$

Proof Assume $\gamma\ \in\ F \cong I\!\!I\oplus F$. Since γ is a bijection by assumption and catamorphism construction preserves bijections it follows that $(\!|I\oplus;\ \gamma|\!)$ is a bijection. Thus, by the constructive Knaster-Tarski theorem (theorem 41) it suffices to prove that $(\!|I\oplus;\ \gamma|\!)$ is surjective. Now,

$$
\begin{array}{ll}
 & (\!|I\oplus;\ \gamma|\!)< \ =\ F.I \\
\equiv & \qquad \{ \qquad \gamma \text{ is an imp } \quad \} \\
 & (\!|I\oplus;\ \gamma|\!) \circ (\!|I\oplus;\ \gamma|\!)^{\cup} \ =\ F.I \\
\Leftarrow & \qquad \{ \qquad (\!|I\oplus;\ \gamma|\!) \circ (\!|I\oplus;\ \gamma|\!)^{\cup} \text{ solves the equation} \\
 & \qquad\qquad X ::\qquad X\ =\ \gamma \circ I\oplus X \circ \gamma^{\cup}\ , \\
 & \qquad\qquad \gamma^{\cup} \text{ is } (I\oplus)\text{-reductive.} \quad \} \\
 & F.I\ =\ \gamma\ \circ\ I \oplus F.I\ \circ\ \gamma^{\cup} \\
\equiv & \qquad \{ \qquad \gamma\ \in\ F \cong I\!\!I\oplus F \quad \} \\
 & \mathit{true}\ .
\end{array}
$$

□

Theorem 45 (Constructive Fusion) Suppose that $\oplus$ and $\otimes$ are binary relators. Suppose that $\oslash$ is also a binary relator but such that $(\oslash I)$ is universally $\sqcup$-junctive. Then

$$
\begin{array}{ll}
 & (\!|I\oplus;\ \gamma|\!)\ \in\ [\mu(S \mapsto R\otimes S) \oslash R]\ \cong\ [\mu(S \mapsto R\oplus S)] \\
\Leftarrow & \gamma\ \in\ [(R\otimes S)\oslash R]\ \cong\ [R\oplus(S\oslash R)]\ \ .
\end{array}
$$

Proof Assume that $\gamma\ \in\ [(R\otimes S)\oslash R]\ \cong\ [R\oplus(S\oslash R)]$. We begin by noting that

$$
\begin{array}{ll}
 & \gamma\ \in\ [(R\otimes S)\oslash R]\ \cong\ [R\oplus(S\oslash R)] \\
\equiv & \qquad \{ \qquad \text{binary relators: definition 22} \quad \} \\
 & \quad \gamma\ \in\ [(I\otimes S)\oslash I]\ \cong\ [I\oplus(S\oslash I)] \\
 & \wedge\ \gamma\ \in\ [(R\otimes I)\oslash R]\ \cong\ [R\oplus(I\oslash R)] \\
\equiv & \qquad \{ \qquad \text{definition} \quad \} \\
 & \quad \gamma \in (\oslash I\ \cdot\ I\otimes) \cong (I\oplus\ \cdot\ \oslash I) \\
 & \wedge\ \gamma\ \in\ [(R\otimes I)\oslash R]\ \cong\ [R\oplus(I\oslash R)]\ \ .
\end{array}
$$

We now try to find a candidate isomorphism using theorem 44. We have, for all specs β:

$$
\begin{array}{ll}
& \llparenthesis I\oplus;\ \gamma \rrparenthesis \ \in\ [\mu(S \mapsto R{\otimes}S) \oslash R] \ \cong\ [\mu(S \mapsto R{\oplus}S)] \\
\Leftarrow & \qquad \{ \qquad \text{theorem 44} \quad \} \\
& \quad \beta \ \in\ [\mu(S \mapsto R{\otimes}S) \oslash R] \ \cong\ [R\oplus(\mu(S \mapsto R{\otimes}S) \oslash R)] \\
& \wedge\ \beta^{\cup} \text{ is } (I\oplus)\text{-reductive} \\
& \wedge\ \llparenthesis I\oplus;\ \gamma \rrparenthesis = \llparenthesis I\oplus;\ \beta \rrparenthesis \ \ .
\end{array}
$$

The proof now splits into three parts. First we construct an isomorphism β of the required type. Then we verify that $\beta^{\cup}$ is $I\oplus$-reductive. Finally we verify that $\llparenthesis I\oplus;\ \gamma \rrparenthesis = \llparenthesis I\oplus;\ \beta \rrparenthesis$.

For brevity let us introduce the notation $\ddagger.R$ for $\mu(S \mapsto R{\otimes}S)$. Then our first requirement translates into the construction of β such that

$$
\beta \ \in\ \ddagger{\oslash}\mathbb{I} \cong \mathbb{I}\oplus(\ddagger{\oslash}\mathbb{I}) \ \ .
$$

This is achieved by restricting the left domain of γ to $\ddagger.I{\oslash}I$. Specifically, we have:

$$
\ddagger.I{\oslash}I \ \circ\ \gamma \quad\in\quad \ddagger{\oslash}\mathbb{I} \cong \mathbb{I}\oplus(\ddagger{\oslash}\mathbb{I}) \ \ .
$$

The proof is straightforward. One has to verify three conditions, namely:

$$
(\ddagger.I{\oslash}I \ \circ\ \gamma)< \ =\ \ddagger.I{\oslash}I \ \ ,
$$

$$
\ddagger.R{\oslash}R \ \circ\ \gamma \quad=\quad \gamma \circ R\oplus(\ddagger.R{\oslash}R) \ \ , \text{for all } R, \text{ and}
$$

$$
(\ddagger.I{\oslash}I \ \circ\ \gamma)> = I\oplus(\ddagger.I{\oslash}I) \ \ .
$$

This is left to the reader.

For the second part we have to show that $(\ddagger.I{\oslash}I \ \circ\ \gamma)^{\cup}$ is $(I\oplus)$-reductive. We prove a slightly more general result, namely, for all universally $\sqcup$-junctive relators G, all specs δ, and all relators H,

$$
\delta \ \in\ G\cdot(I\otimes) \cong (H\cdot G) \quad\Rightarrow\quad (G.\ddagger.I \ \circ\ \delta)^{\cup} \text{ is } H\text{-reductive.}
$$

The latter is a consequence of the reductivity translation theorem in [8] and the fact that $\ddagger.I$ is $(I\otimes)$-reductive. (Make the substitution $F := (I\otimes)$ in the statement of the theorem.) Now combining this with our initial observation (specifically

$$
\gamma \ \in\ [(R{\otimes}S){\oslash}R] \ \cong\ [R\oplus(S{\oslash}R)] \quad\Rightarrow\quad \gamma \ \in\ ({\oslash}I)\cdot(I\otimes) \ \cong\ (I\oplus)\cdot({\oslash}I)
$$

) using the substitutions $G := ({\oslash}I)$ and $H := (I\oplus)$ we obtain

$$
(\ddagger.I{\oslash}I \ \circ\ \gamma)^{\cup} \ \text{ is } (I\oplus)\text{-reductive}
$$

as required.

The third, and final step, is to verify that

$$
\llparenthesis I\oplus;\ \gamma \rrparenthesis \ =\ \llparenthesis I\oplus;\ \ddagger.I \oslash I \ \circ\ \gamma \rrparenthesis \ \ .
$$

This follows since:

$$
\begin{array}{ll}
& (\!|I\oplus;\ \gamma|\!) \ = \ (\!|I\oplus;\ \ddagger.I \oslash I \ \circ\ \gamma|\!) \\
\Leftarrow & \qquad \{ \qquad \text{domain trading} \quad \} \\
& \ddagger.I \oslash I \ \sqsupseteq \ (\gamma \circ I\oplus(\ddagger.I \oslash I))< \\
\equiv & \qquad \{ \qquad \gamma \ \in \ (\oslash I)\cdot(I\otimes) \cong (I\oplus)\cdot(\oslash I) \quad \} \\
& \ddagger.I \oslash I \ \sqsupseteq \ ((I \otimes \ddagger.I)\oslash I \circ \gamma)< \\
\equiv & \qquad \{ \qquad \ddagger.I \ = \ I \otimes \ddagger.I \text{ , domains} \quad \} \\
& \mathit{true} \quad .
\end{array}
$$

□

Corollary 46 Suppose that G is a universally $\sqcup$-junctive relator. Then

$$
\begin{array}{ll}
& (\!|I\oplus;\ \gamma|\!) \ \in \ [G.\mu(S \mapsto R\otimes S)] \ \cong \ [\mu(S \mapsto R\oplus S)] \\
\Leftarrow & \gamma \ \in \ [G.(R\otimes S)] \ \cong \ [R \oplus G.S] \quad .
\end{array}
$$

Proof Define the binary relator $\oslash$ by $S\oslash R = G.S$ for all R and S. That it is a relator is obvious. It is also obvious that $(\oslash I)$ is universally $\sqcup$-junctive. It remains to instantiate theorem 45 with the above definition of $\oslash$.

□

The identity relator is trivially universally $\sqcup$-junctive. So, as a special case of corollary 46, we also have:

Corollary 47

$$
(\!|I\oplus;\ \gamma|\!) \ \in \ [\mu(S \mapsto R\otimes S)] \ \cong \ [\mu(S \mapsto R\oplus S)] \ \Leftarrow \ \gamma \in \otimes \cong \oplus \quad .
$$

□

4.3 True versus Isomorphic Fixed Points

In the spec calculus, fixed points are *true* fixed points and not up to isomorphism as is the case in formal systems based on category theory (or equivalents). This has major advantages for economy of calculation, which advantages we exploit extensively. The main advantage is that, of the lattice-theoretic fixed point rules, we have found it necessary to give constructive forms to just the laws (25) and (27): in our calculus the fixed point equation (24), the rolling rule (26), and the diagonal rule (28) remain equations and do not have to be rewritten as isomorphisms. In other systems, as we have said, this is not the case. Those working in category-based formal systems may thus find it necessary to identify and prove the constructive forms of (24), (26) and (28). (This shouldn't be too difficult.)

A simple example helps to illustrate this. Consider the datatype (cons) list. In the spec calculus this would be defined as the relator $*$ where

$$
*R = \mu(X \mapsto \mathbb{1}+(R\times X)) \quad .
$$

By application of the rolling rule (26), with f instantiated to the function $(R\times)$, and g instantiated to the function $(\mathbb{1}+)$ we have

(48) $\quad R \times *R \;=\; \mu(X \mapsto R\times(\mathbb{1}+X)) \quad .$

This *equality* in the spec calculus is an *isomorphism* in conventional programming languages. In a programming language such as ML, Gofer, Haskell etc. we can encode two datatypes L and T (with the aid of an auxilliary datatype S) as follows:

```
data L a = Nill | Cons (a) (L  a)
data S a = Nils | Id (T a)
data T a = Pair (a) (S a)
```

The relator $(R \mapsto \mu(X \mapsto R\times(\mathbb{1}+X)))$ corresponds to the datatype T and the relator $*$ corresponds to the datatype L. The datatypes $R \mapsto R \times *R$ and T are, however, unequal and cannot be freely interchanged in a program. It is necessary, in any of the languages mentioned above, to write functions mapping values of the former type to values of type T specifically as follows:

```
toT (x,y) = Pair (x) (f y)

f Nill = Nils
f (Cons u v) = Id (toT (u,v))

fromT (Pair x y) = (x, g y)

g Nils = Nill
g (Id z) = (uncurry Cons) (fromT z)
```

Note that the functions do nothing more than swap labels, specifically $(\_,\_)$ is replaced by *Pair* and vice-versa, *Nill* is replaced by *Nils* and vice-versa, and *Cons* by *Id* and vice-versa.

The identity (48) is one that is used repeatedly later in this paper. Because it and similar identities are only isomorphisms in conventional programming languages the reader may experience problems in implementing the more complex isomorphisms that we derive. One must bear in mind that wherever we appeal to the fixed-point equation (24), rolling rule (26) or the diagonal rule (28) an additional bijection must be constructed.

Aside During and shortly after the preparation of the first draft of this paper (July 1993) Lambert Meertens and Erik Meijer have filled in some of the gaps between our work and category-theory-based formalisms.

Meertens has considered the rolling rule. In outline he gives the following construction. Let F and G be functors and let $(A_{GF}\ ,\ \mathsf{in}_{GF})$, respectively $(A_{FG}\ ,\ \mathsf{in}_{FG})$, be an initial GF algebra, respectively an initial FG algebra. (We use juxtaposition locally to denote composition of functors.) Then $(F.A_{GF}\ ,\ F.\mathsf{in}_{GF})$ is an initial FG algebra and $(\![FG;\ F.\mathsf{in}_{GF}]\!)$ is an isomorphism between $(F.A_{GF}\ ,\ F.\mathsf{in}_{GF})$ and $(A_{FG}\ ,\ \mathsf{in}_{FG})$. Moreover, if $(A\ ,\ \varphi)$ is an FG algebra then $\varphi \circ F.(\![GF;\ G.\varphi]\!)$ is the (unique) FG algebra morphism from $(F.A_{GF}\ ,\ F.\mathsf{in}_{GF})$ to $(A\ ,\ \varphi)$. In particular, $\mathsf{in}_{FG} \circ F.(\![GF;\ G.\mathsf{in}_{FG}]\!)$ is the inverse of $(\![FG;\ F.\mathsf{in}_{GF}]\!)$. This general construction is exemplified by the functions `toT` and `fromT` above.

Meijer states the rolling rule, the diagonal rule and μ-fusion in the context of a CPO (although in the case of the diagonal rule he does not explictly state how to construct the witness). His constructive μ-fusion rule appears to be stronger than the one stated here since he demands only the combination of strictness and continuity where we have demanded universal $\sqcup$-junctivity. (The difference is positively finite $\sqcup$-junctivity.) He does not however give any applications of the μ-fusion rule and I have been unable to discover any applications where the additional strength of his rule is needed.
End of Aside.

5 Monads

As a first illustration of the constructive fixed point theorems we discuss in this section a number of theorems about monads. Although more abstract than the properties of lists discussed in the coming sections, the calculations here are simpler. The two sets of applications are however independent and the reader wishing to skip this section may do so without problem.

5.1 Constructive Closure Operators

In category theory closure operators are cited as instances of monads. We want to take the opposite view: monads are constructive closure operators.

One definition of a closure operator is a monotonic endofunction on a preorder that is both reflexive and idempotent. A first stab at a definition of a monad is thus a relator that is constructively reflexive and constructively idempotent.

A monad is thus a relator F such that there exist two specs unit and mul with

(49) $\mathsf{unit} \in F \gtrsim I\!I$,

(50) $\mathsf{mul} \in F \gtrsim F \cdot F$.

(In category theory the symbols η and μ are often used for unit and mul. The use of μ would, however, cause confusion with our use of it as the least fixed point operator.)

To the requirements (49) and (50) we add two coherence conditions, motivated as follows.

Note that the combination of these two postulates with the monotonicity of composition means that monad F simulates F^i for all $i \geq 0$ (where, by definition, $F^0 = I\!I$ and $F^{i+1} = F \cdot F^i$). For example, the calculation

$$
\begin{array}{ll}
 & F \gtrsim F \cdot F \cdot F \\
\Leftarrow & \{ \quad \text{transitivity} \quad \} \\
 & F \gtrsim F \cdot F \quad \wedge \quad F \cdot F \gtrsim F \cdot F \cdot F \\
\Leftarrow & \{ \quad \text{monotonicity} \quad \} \\
 & F \gtrsim F \cdot F \quad \wedge \quad F \gtrsim F \cdot F \quad \wedge \quad F \gtrsim F
\end{array}
$$

can be lifted to the construction of a simulation as follows:

$$
\begin{array}{ll}
& \alpha \in F \gtrsim F \cdot F \cdot F \\
\Leftarrow & \{ \quad \bullet \quad \alpha = \beta \circ \gamma \ , \quad \} \\
& \beta \in F \gtrsim F \cdot F \quad \wedge \quad \gamma \in F \cdot F \gtrsim F \cdot F \cdot F \\
\Leftarrow & \{ \quad \bullet \quad \gamma = \delta \circ F.F.\eta \ , \quad \} \\
& \beta \in F \gtrsim F \cdot F \quad \wedge \quad \delta \in F \gtrsim F \cdot F \quad \wedge \quad \eta \in F \gtrsim F \quad .
\end{array}
$$

Thus assuming (49) and (50)

$$\mathsf{mul} \circ \mathsf{mul} \circ F.F.F.I \quad \in \quad F \gtrsim F \cdot F \cdot F \quad .$$

Simplifying, (since $\mathsf{mul} \circ F.F.F.I = F.F.I \circ \mathsf{mul}$ and $\mathsf{mul} \circ F.F.I = \mathsf{mul}$)

$$\mathsf{mul} \circ \mathsf{mul} \quad \in \quad F \gtrsim F \cdot F \cdot F \quad .$$

This however is not the only way to construct a witness. In the second step $F \cdot F \cdot F$ was parsed as $(F \cdot F) \cdot F$ in order to apply the monotonicity rule. Had we parsed it as $(F \cdot F) \cdot F$ then we would have deduced

$$\mathsf{mul} \circ F.\mathsf{mul} \quad \in \quad F \gtrsim F \cdot F \cdot F \quad .$$

Similarly there are three obvious ways of constructing a simulation of F by itself. First, $F.I$ witnesses such a simulation. Second, since $F = \mathbb{I} \cdot F$, $\mathsf{mul} \circ \mathsf{unit}$ witnesses such a simulation. Third, since $F = F \cdot \mathbb{I}$, $\mathsf{mul} \circ F.\mathsf{unit}$ witnesses the simulation. In order to guarantee unicity of simulations of F^i by F for each i , two *coherence conditions* — the so-called *associative* and *unit* laws — are added to the definition. In full the definition of a monad thus becomes:

Definition 51 (Monad) A monad is a triple $(F \ , \mathsf{unit} \ , \mathsf{mul})$ such that F is a relator and unit and mul are two specs with

(a) $\quad \mathsf{unit} \in F \gtrsim \mathbb{I}$,
(b) $\quad \mathsf{mul} \in F \gtrsim F \cdot F$,
(c) $\quad \mathsf{mul} \circ \mathsf{mul} = \mathsf{mul} \circ F.\mathsf{mul}$,
(d) $\quad \mathsf{mul} \circ \mathsf{unit} = F.I = \mathsf{mul} \circ F.\mathsf{unit}$.

(Property (c) is called the associative law and (d) the unit law.)

□

Having identified a class of relators the first test that it should undergo is whether or not it is closed under natural isomorphisms. For monads we have:

Theorem 52 Suppose $(F \ , \mathsf{unit} \ , \mathsf{mul})$ is a monad and γ is a bijection with $\gamma< = F.I$. Then $(F^\gamma \ , (\gamma^\cup \circ \mathsf{unit}) \ , (\gamma^\cup \circ \mathsf{mul} \circ F.\gamma \circ \gamma))$ is a monad where $F^\gamma.R = \gamma^\cup \circ F.R \circ \gamma$ for all R.

Proof By definition, $\gamma \in F \cong F^\gamma$ and hence $\gamma \in F \gtrsim F^\gamma$ and $\gamma^\cup \in F^\gamma \gtrsim F$. The construction of the unit is a straightforward application of transitivity. The construction of the multiplier is obtained by adding witnesses to the following calculation:

$$
\begin{array}{ll}
 & F^\gamma \gtrsim F^\gamma \cdot F^\gamma \\
\Leftarrow & \{ \quad \text{transitivity and monotonicity} \quad \} \\
 & F^\gamma \gtrsim F \cdot F \quad \wedge \quad F \gtrsim F^\gamma \\
\Leftarrow & \{ \quad \text{transitivity} \quad \} \\
 & F^\gamma \gtrsim F \quad \wedge \quad F \gtrsim F \cdot F \quad \wedge \quad F \gtrsim F^\gamma \ .
\end{array}
$$

This leads to the witness

$$\gamma^\cup \circ \mathsf{mul} \circ \gamma \circ F^\gamma.\gamma$$

which is equal to

$$\gamma^\cup \circ \mathsf{mul} \circ F.\gamma \circ \gamma \ .$$

Verification of 51(c) and 51(d) is tedious but straightforward.
□

5.2 A Class of Monads

A central theorem in the theory of closure operators is the following: let $(\mathcal{L}, \sqsubseteq)$ be a complete lattice, let $(\mathcal{M}.\mathcal{L}\ ,\ \sqsubseteq)$ be the (complete) lattice of monotonic endofunctions on $\mathcal{L}$, and let $(\mathcal{C}.\mathcal{L}\ ,\ \sqsubseteq)$ be the complete lattice of closure operators on $\mathcal{L}$ whereby the ordering $\sqsubseteq$ on functions is the usual pointwise extension of the ordering $\sqsubseteq$ on elements of $\mathcal{L}$. Then the function $\star$ defined by

$$f^\star.x \ = \ \mu(y \mapsto (x \sqcup f.y))$$

is a (total, surjective) function from $\mathcal{M}.\mathcal{L}$ to $\mathcal{C}.\mathcal{L}$. (It is indeed the lower (Galois) adjoint of the function embedding elements of $\mathcal{C}.\mathcal{L}$ in $\mathcal{M}.\mathcal{L}$ although we will not make use of this fact here.)

The $\star$ notation has been introduced by the author to denote this function in order to make the link with the Kleene star used in later sections. The point is that the closure star has many properties in common with the Kleene star. (But beware, not all properties of the Kleene star are enjoyed by the closure star. It is not, for instance, the case in general that $f \cdot f^\star = f^\star \cdot f$.) In particular, the closure star satisfies the decomposition property

$$(f \sqcup g)^\star \ = \ g^\star \cdot (f \cdot g^\star)^\star \ .$$

In this section we define a function $\star$ on relators by analogy with the definition of $\star$ on monotonic functions. We then show, via a *monad simulation* theorem that, for all relators F, $F^\star$ is a monad. Furthermore, in the next subsection we establish a "constructive " closure decomposition theorem, i.e. a monad decomposition theorem.

Let F be an arbitrary relator. Define the unary relator $F^\star$ by

$$F^\star.X \;=\; \mu(Y \;\mapsto\; X + F.Y) \;\;.$$

Define in addition the specs τ_F and η_F by

(53) $$\tau_F \;=\; \hookrightarrow \;=\; F^\star.I \circ \hookrightarrow \;,$$

(54) $$\eta_F \;=\; \hookleftarrow \circ F.F^\star.I \;=\; F^\star.I \circ \hookleftarrow \;.$$

Examples of relators defined in this way are discussed in combination with examples of the monad decomposition theorem in subsection 5.5. Our goal in this section is to establish that the relator $F^\star$ can be furnished with a unit and a multiplier .

We first observe that τ_F and η_F witness simulations:

(55) $$\tau_F \;\in\; F^\star \gtrsim I\!I \;\;,$$

(56) $$\eta_F \;\in\; F^\star \gtrsim F \cdot F^\star \;\;.$$

Thus τ_F is an appropriate choice for the unit. To see how to construct the multiplier we first apply the constructive Knaster-Tarski theorem in order to observe a general monad simulation theorem: Define the binary relator $+_F$ by

$$X \;+_F\; Y \;=\; X + F.Y \;\;.$$

Also, abbreviate $(\!|(I{+_F});\; R|\!)$ to $(\!|R|\!)_F$. Then we have:

Theorem 57 (Monad Simulation) For all relators F, G and H, and all specs R,

$$(\!|R|\!)_F \;\in\; H \gtrsim F^\star \cdot G \;\;\Leftarrow\;\; R \;\in\; H \gtrsim G + F \cdot H \;\;.$$

Proof

$$\begin{array}{ll}
 & (\!|R|\!)_F \;\in\; H \gtrsim F^\star \cdot G \\
\equiv & \quad \{ \quad (F^\star \cdot G).R \;=\; F^\star.(G.R) \;=\; \mu(X \;\mapsto\; G.R + F.X) \quad \} \\
 & (\!|R|\!)_F \;\in\; H \gtrsim [\mu(X \;\mapsto\; G.R + F.X)] \\
\Leftarrow & \quad \{ \quad \text{constructive Knaster-Tarski: 41} \\
 & \qquad\quad \text{with } \oplus \text{ defined by } R{\oplus}X \;=\; G.R + F.X \quad \} \\
 & R \in H \gtrsim [G.R + F.H.R] \;\;\wedge\;\; (\!|R|\!)_F \;=\; (\!|X \mapsto G.I{+}F.X;\; R|\!) \\
\equiv & \quad \{ \quad \text{definition} \quad \} \\
 & R \;\in\; H \gtrsim G + F \cdot H \;\;\wedge\;\; (\!|R|\!)_F \;=\; (\!|X \mapsto G.I{+}F.X;\; R|\!) \;\;.
\end{array}$$

To complete the proof we have to establish that $(\!|R|\!)_F \;=\; (\!|X \mapsto G.I{+}F.X;\; R|\!)$. This we do as follows:

$$
\begin{aligned}
& (\!|X \mapsto G.I{+}F.X;\ R|\!) \\
= \quad & \{ \quad (36) \quad \} \\
& (\!|X \mapsto I{+}F.X;\ R|\!) \circ (\!|X \mapsto I{+}F.X;\ G.I|\!) \\
= \quad & \{ \quad (35) \quad \} \\
& (\!|X \mapsto I{+}F.X;\ R \circ G.I{+}I|\!) \\
= \quad & \{ \quad \bullet \quad R \in H \gtrsim G + F \cdot H \text{ , definition of } (\!|R|\!)_F \quad \} \\
& (\!|R|\!)_F \ .
\end{aligned}
$$

□

With the aid of the monad simulation theorem we can construct the multiplier and verify that we have a monad.

Theorem 58 The triple $(F^\star\ , \tau_F\ , (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F)$ is a monad.

Proof First we construct $(\!|F^\star.I \ \triangledown\ \eta_F|\!)_F$ as a witness of a simulation of $F^\star \cdot F^\star$ by $F^\star$:

$$
\begin{aligned}
& \alpha \in F^\star \gtrsim F^\star \cdot F^\star \\
\Leftarrow \quad & \{ \quad \text{monad simulation: 57} \\
& \quad \bullet \quad \alpha = (\!|\beta|\!)_F \ , \quad \} \\
& \beta \in F^\star \gtrsim F^\star + F \cdot F^\star \\
\Leftarrow \quad & \{ \quad \bullet \quad \beta = F^\star.I \ \triangledown\ \gamma \text{ , theorem 23} \quad \} \\
& \gamma \in F^\star \gtrsim F \cdot F^\star \\
\Leftarrow \quad & \{ \quad (56) \quad \} \\
& \gamma = \eta_F \ .
\end{aligned}
$$

Now we have to verify the two coherence conditions. We show how to verify the associative law in order to demonstrate the technique and leave the unit law to the reader.

$$
\begin{aligned}
& (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \circ F^\star.(\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \\
= \quad & \{ \quad \text{map fusion: (35)} \quad \} \\
& (\!|(F^\star.I \ \triangledown\ \eta_F) \circ ((\!|F^\star.I \ \triangledown\ \eta_F|\!)_F + F.I)|\!)_F \\
= \quad & \{ \quad \text{junc-sum fusion,} \quad \eta_F \circ F.I = \eta_F \ , \\
& \quad F^\star.I \circ (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F = (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \quad \} \\
& (\!|(\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \ \triangledown\ \eta_F|\!)_F \ .
\end{aligned}
$$

Hence,

$$
\begin{aligned}
& (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \circ (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \\
= \ & (\!|F^\star.I \ \triangledown\ \eta_F|\!)_F \circ F^\star.(\!|F^\star.I \ \triangledown\ \eta_F|\!)_F
\end{aligned}
$$

$$
\begin{array}{ll}
\Leftarrow & \{ \quad \text{catamorphism fusion: (34), and above} \quad \} \\
& (\!|F^\star.I \;\triangledown\; \eta_F|\!)_F \circ (F^\star.I \;\triangledown\; \eta_F) \\
& = ((\!|F^\star.I \;\triangledown\; \eta_F|\!)_F \;\triangledown\; \eta_F) \circ (I + F.(\!|F^\star.I \;\triangledown\; \eta_F|\!)_F) \\
\equiv & \{ \quad \text{junc-sum fusion, junc-sum cancellation} \quad \} \\
& (\!|F^\star.I \;\triangledown\; \eta_F|\!)_F \circ F^\star.I \;=\; (\!|F^\star.I \;\triangledown\; \eta_F|\!)_F \circ I \\
& \wedge\; (\!|F^\star.I \;\triangledown\; \eta_F|\!)_F \circ \eta_F \;=\; \eta_F \circ F.(\!|F^\star.I \;\triangledown\; \eta_F|\!)_F \\
\equiv & \{ \quad \text{domain of catamorphisms and computation rules} \quad \} \\
& \mathit{true} \quad .
\end{array}
$$

□

For monotype A the monotype $F^\star.A$ is referred to as the *free F-algebra generated by A*; $F^\star.\perp\!\!\perp$ is an *initial F-algebra*, it being isomorphic to μF . (Compare with $f^\star.\perp\!\!\perp \;=\; \mu f$ in lattice theory.)

5.3 The Decomposition Theorem

We are now in a position to prove a constructive closure decomposition theorem, in other words a *monad decomposition theorem*. In section 5.5 we give examples of its use.

In fact, the monad decomposition theorem has a very straightforward proof in the spec calculus as we will now demonstrate. Unfortunately, however, this proof is likely to be rejected by functional programmers for reasons to be explained after the proof has been given. In section 5.4, therefore, we outline a rather more complicated proof of the theorem which should be acceptable to users of conventional programming languages.

Theorem 59 (Monad Decomposition) For all relators F and G,

$$G^\star \cdot (F \cdot G^\star)^\star \;=\; (F{+}G)^\star \quad .$$

Proof We first observe the equality:

$$(G^\star \cdot (F \cdot G^\star)^\star).R \;=\; \mu(X \mapsto (R + F.X) + G.X)$$

for all specs R.

$$
\begin{array}{ll}
& (G^\star \cdot (F \cdot G^\star)^\star).R \\
= & \{ \quad \text{function composition, definition of } {}^\star \quad \} \\
& G^\star \,.\, \mu(X \mapsto R + (F \cdot G^\star).X) \\
= & \{ \quad \text{rolling rule: (26)} \\
& \qquad f{,}g \;:=\; G^\star \,,\, (X \mapsto R + F.X) \quad \} \\
& \mu(X \mapsto G^\star \,.\, (R + F.X)) \\
= & \{ \quad \text{definition of } {}^\star \quad \} \\
& \mu(X \mapsto \mu(Y \mapsto (R + F.X) + G.Y))
\end{array}
$$

$$= \qquad \{ \quad \text{diagonal rule: (28)} \quad \}$$

$$\mu(X \mapsto (R + F.X) + G.X) \ .$$

The proof of the decomposition theorem is now a straightforward application of theorem 47. Let sumass denote the isomorphism between the ternary relators $[(R{+}S){+}T]$ and $[R{+}(S{+}T)]$. Then we have

$$\mathsf{sumass} \circ I + (F.I + G.I)$$

$$\in \quad [(R + F.X) + G.X] \cong [R + (F.X + G.X)] \ .$$

Thus, by the above-mentioned theorem together with domain trading,

$$(\!\![X \mapsto I{+}(F.X + G.X);\ \mathsf{sumass}]\!\!) \ \in \ G^\star \cdot (F \cdot G^\star)^\star \ \cong \ (F{+}G)^\star \ .$$

Its inverse, by (31), is

$$(\!\![X \mapsto (I + F.X){+}G.X;\ \mathsf{sumass}^\cup]\!\!) \ \in \ (F{+}G)^\star \ \cong \ G^\star \cdot (F \cdot G^\star)^\star \ .$$

□

5.4 Alternative Implementation

The proof just given of the monad decomposition theorem, being completely equational, is concise and leads to a concise program for the witness to the isomorphism and its inverse. Unfortunately, it is unlikely that any existing functional programming language would accept the witness on typing grounds. Certainly the inverse would be rejected on those grounds.

An alternative to the above calculation, that does lead to witnesses acceptable to conventional compilers, is to construct two simulations, one of type $G^\star \cdot (F \cdot G^\star)^\star \gtrsim (F{+}G)^\star$ and the other of type $(F{+}G)^\star \gtrsim G^\star \cdot (F \cdot G^\star)^\star$, and then show that they are inverses of each other. The construction of the simulations is not difficult but nevertheless instructive: the basic tool is the monad simulation theorem, and it is a straightforward exercise for anyone familiar with regular algebra (as most well-trained computing scientists are). Proving that the two witnesses are inverses of each other is however an arduous and non-trivial task. (See [4] for details.)

For those who wish to carry out the exercise, or program the solution in their favourite programming language, we present the solution below. For brevity we define the relator K by

$$K \ = \ F \cdot G^\star \ .$$

We also use the notations τ_F and η_F as defined by (53) and (54) (for each relator F).

The witness to the simulation of type $G^\star \cdot (F \cdot G^\star)^\star \gtrsim (F{+}G)^\star$ is

$$(\!\![(\tau_F \circ \tau_K) \ \triangledown \ ((\tau_G \circ \eta_K) \ \triangledown \ \eta_G)]\!\!)_{F+G} \ .$$

Its inverse, of type $(F{+}G)^\star \gtrsim G^\star \cdot (F \cdot G^\star)^\star$, is defined by T_0 where

$$T_0 \ = \ (\!\![T_1 \ \triangledown \ (\eta_{F+G} \circ \hookleftarrow)]\!\!)_G \ ,$$

$$T_1 \ = \ (\!\![\tau_{F+G} \ \triangledown \ (\eta_{F+G} \circ \hookrightarrow \circ F.T_2)]\!\!)_K \ ,$$

$$T_2 \ = \ (\!\![I \ \triangledown \ (\eta_{F+G} \circ \hookleftarrow)]\!\!)_G \ .$$

5.5 Examples

In this section we give some examples of free monads and the application of the monad decomposition theorem.

The simplest examples are obtained by taking F to be a constant relator. Suppose A is a monotype, and let $\hat{A}$ denote the relator $X{\mapsto}A$. Then

$$
\begin{array}{ll}
 & \hat{A}^\star \\
= & \{ \quad F^\star . R \;=\; \mu(X \;\mapsto\; R + F.X) \;=\; R \;+\; F . F^\star . R \\
 & \quad \text{i.e.} \;\; F^\star \;=\; I\!I \;+\; F \cdot F^\star \quad \} \\
 & I\!I \;+\; \hat{A} \cdot \hat{A}^\star \\
= & \{ \quad \hat{A} \cdot F \;=\; \hat{A} \quad \} \\
 & I\!I + \hat{A} \;\; .
\end{array}
$$

Taking A to be $\perp\!\!\perp$ we have in particular that $\hat{\perp\!\!\perp}^\star \;=\; I\!I + \hat{\perp\!\!\perp}$. Since $\hat{\perp\!\!\perp}^\star \;=\; I\!I + \hat{\perp\!\!\perp}$ is isomorphic to $I\!I$ it follows that the identity relator is a monad (as might be expected!).

From the above the unit of $\hat{A}^\star$ is $\hookrightarrow$ and the multiplier is $\hookrightarrow\!\cup + A$. Wadler [32] defines monads not in terms of the unit, multiplier and relator but rather in terms of the unit and a function called "bind". This function (as indeed explained by Wadler) is easily defined in terms of the unit and multiplier. Specifically, for arbitrary monad F,

$$\mathsf{bind}_F.R \;=\; \mathsf{mul}_F \circ F.R \;\; .$$

In particular, for $M \;=\; \hat{A}^\star$,

$$\mathsf{bind}_M.R \;=\; (\hookrightarrow\!\cup \circ R) + A \;\; .$$

This example can also be used to illustrate the monad decomposition theorem. We have:

$$
\begin{array}{ll}
 & (\hat{A}{+}G)^\star \\
\cong & \{ \quad \text{monad decomposition} \quad \} \\
 & G^\star \cdot (\hat{A} \cdot G^\star)^\star \\
= & \{ \quad \hat{A} \cdot F \;=\; \hat{A} \quad \} \\
 & G^\star \cdot \hat{A}^\star \\
= & \{ \quad \hat{A}^\star \;=\; I\!I + \hat{A} \quad \} \\
 & G^\star \;\cdot\; I\!I + \hat{A} \;\; .
\end{array}
$$

The example is actually very well known in a specific form, although it does not appear to be well known that it is an instance of the decomposition theorem. The specific form is this: Take G to be the squaring relator. I.e. define $G.R \;=\; R{\times}R$. Take A to be the unit type $1\!\!1$. Denote $(\hat{1\!\!1}{+}G)^\star.R$ by R^* and $G^\star.R$ by R^+. Then the theorem says that $[(R{+}1\!\!1)^+] \cong [R^*]$. The equivalent theorem at lattice level is that the transitive closure of the

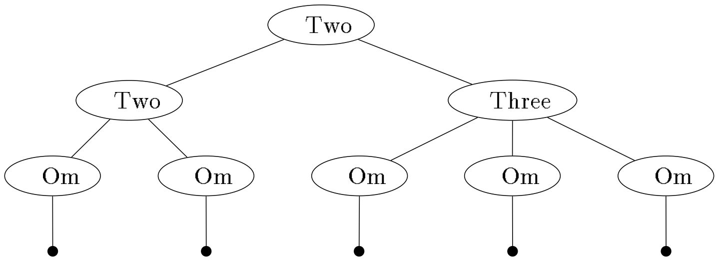

Figure 1: Element of Type M

relation $R{+}1\!\!1$ (where $1\!\!1$ denotes the identity relation and $+$ denotes the union operator) is the reflexive, transitive closure of relation R. Wadler [32] considers this example (with A instantiated to "*Exception*") but does not seem to have been aware of the relevance of monad decomposition.

A second example of the monad decomposition theorem is as follows. Suppose we define four datatypes as follows:

```
data M a  =  Om a | Two (M a) (M a) | Three (M a) (M a) (M a)
data N a  =  On a | Drie (N a) (N a) (N a)
data P a  =  Twee (N a) (N a)
data Q a  =  Oq a | P (Q a)
```

("Twee" and "drie" are the Dutch names for "two" and "three".) In terms of the spec calculus,

$$\begin{aligned} M &= (\mathit{Sq}{+}\mathit{Cube})^\star \quad , \\ N &= \mathit{Cube}^\star \quad , \\ P &= \mathit{Sq}\cdot N \quad , \end{aligned}$$

where

$$\begin{aligned} \mathit{Cube}.R &= R{\times}R{\times}R \quad \text{, and} \\ \mathit{Sq}.R &= R{\times}R \quad . \end{aligned}$$

Thus by the monad decomposition theorem,

$$M \cong N\cdot Q \quad .$$

An element of type M is depicted in fig.1. The corresponding element of type $N\cdot Q$ is shown in fig.2. The construction of one tree from the other is very straightforward: replace the constructor "`Om`" everywhere by "`On Oq`", the constructor "`Two`" by "`On Twee`"and the constructor "`Three`" by "`Drie`" .

6 Applications to Lists

In this section we consider applications of the constructive fixed point theorems that are specific to lists. Classes of isomorphisms with list isomorphisms as special cases are discussed in subsequent sections.

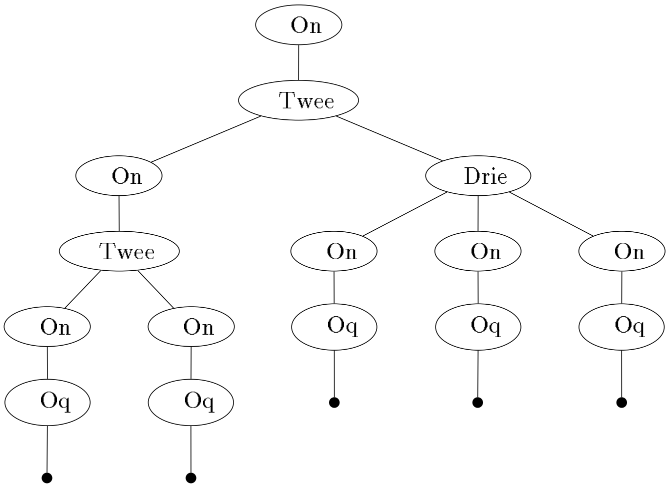

Figure 2: Element of Type $N \bullet Q$

Lists can be defined in various (isomorphic) ways. In the examples that follow we use "cons" lists. That is, we assume the relator $*$ is defined by

$$*R = \mu(X \mapsto \mathbb{1}+(R{\times}X)) \ \ .$$

By unfolding the fixed-point operator we have:

$$*I \quad = \quad \mathbb{1} + (I \times *I) \quad = \quad (\hookrightarrow \circ \mathbb{1}) \triangledown (\hookleftarrow \ \circ \ I \times *I) \ \ .$$

Letting nil denote $\hookrightarrow \circ \mathbb{1}$ and cons denote $\hookleftarrow \ \circ \ I \times *I$ we thus have:

$$*I \quad = \quad \mathsf{nil} \triangledown \mathsf{cons} \ \ .$$

In words, the datatype list has two constructors nil and cons. (In category-based systems this equality is an isomorphism.) Moreover, the two constructors are polymorphic in the sense that

(60) $\quad \mathsf{nil} \in [*R] \overset{>}{\sim} \hat{\mathbb{1}} \ \ ,$

and

(61) $\quad \mathsf{cons} \in [*R] \overset{>}{\sim} [R \times *R] \ \ .$

In the applications the only knowledge we need of the polynomial relators is that both $(\times I)$ and $(I\times)$ are universally $\sqcup$-junctive, and — up to isomorphism — they all obey the usual arithmetic laws. Specifically we assume the existence of the following natural isomorphisms:

$$\mathsf{lunit} \in [\mathbb{1}{\times}R] \cong [R] \ \ ,$$

$$\mathsf{runit} \in [R{\times}\mathbb{1}] \cong [R] \ \ ,$$

$$\mathsf{sumass} \in [(R{+}S){+}T] \cong [R{+}(S{+}T)] \ ,$$

$$\mathsf{proass} \in [(R{\times}S){\times}T] \cong [R{\times}(S{\times}T)] \ ,$$

$$\mathsf{rdist} \in [(R{+}S){\times}T] \cong [(R{\times}T){+}(S{\times}T)] \ ,$$

$$\mathsf{ldist} \in [T{\times}(R{+}S)] \cong [(T{\times}R){+}(T{\times}S)] \ .$$

(The existence of these isomorphisms is, of course, well known. For details of how to construct them in the spec calculus see [5].)

Note that nowhere in our calculations do we assume that $\times$ is commutative. In regular algebra this is not the case, and in order to properly model the theory of lists it is vital that we do not do so.

Note also that addition is *not* idempotent, although this *is* the case in regular algebra. Identities in regular algebra that rely on this property are not constructively true. For example the $*$ operator is idempotent in regular algebra, whereas it is not the case that the datatype list of lists is isomorphic to the datatype list .

The first example is chosen for its simplicity and because it is fundamental to other examples.

In this and subsequent examples we employ a notation whereby the witnesses to isomorphisms are included in the hints marked by a bullet ("•"). Specifically a proof step of the form

$$\begin{array}{ll} & F \\ \cong & \{ \quad \bullet \ \ \alpha \ , \\ & \qquad \text{hint} \quad \} \\ & G \end{array}$$

is short for

$$\alpha \in F \cong G \ \Leftarrow \ \text{hint} \ .$$

Theorem 62 (List Fusion)

$$[*R \times S] \cong [\mu(X \mapsto S{+}(R{\times}X))] \ .$$

Proof We construct a witness α as follows:

$$\begin{array}{ll} & \alpha \in [*R \times S] \cong [\mu(X \mapsto S{+}(R{\times}X))] \\ \equiv & \{ \quad \text{definition of } * \quad \} \\ & \alpha \in [\mu(X \mapsto \mathbb{1}{+}(R{\times}X)) \times S] \cong [\mu(X \mapsto S{+}(R{\times}X))] \\ \Leftarrow & \{ \quad \text{constructive fusion: theorem 45} \end{array}$$

$$\bullet \quad \alpha = (\![X \ \mapsto \ I{+}(I{\times}X);\ \beta]\!) \ .$$

The relators $\oplus$, $\otimes$ and $\oslash$ in the theorem are instantiated to *ternary* relators viewed as binary relators, one argument being a pair of specs. Specifically, we substitute

$(R,S){\otimes}X \ := \ \mathbb{1}{+}(R{\times}X)$,

$(R,S){\oplus}X \ := \ S{+}(R{\times}X)$, and

$X{\oslash}(R,S) \ := \ X{\times}S$. }

$$
\begin{array}{ll}
 & \beta \ \in \ [(\mathbb{1}{+}(R{\times}X)){\times}S] \ \cong \ [S{+}(R{\times}(X{\times}S))] \\
\Leftarrow & \quad \{ \quad \text{arithmetic} \quad \} \\
 & \beta \ = \ \mathsf{rdist} \circ \mathsf{lunit}{+}\mathsf{proass} \ .
\end{array}
$$

We have thus constructed

$$
\begin{array}{ll}
 & (\![X \ \mapsto \ I{+}(I{\times}X);\ \mathsf{rdist} \circ \mathsf{lunit}{+}\mathsf{proass}]\!) \\
\in & [*R \times S] \cong [\mu(X \mapsto S{+}(R{\times}X))] \ .
\end{array}
$$

$\Box$

For our second example we consider the problem of constructing a function that “joins” or “appends” two (cons) lists. Formulated in terms of simulations the problem is to show that the datatype list simulates the datatype pair of lists.

Theorem 63 (List Append)

$$[*R] \ \gtrsim \ [*R \times *R] \ .$$

Proof By list fusion and monotonicity we have

$$
\begin{array}{ll}
 & [\![X \ \mapsto \ *I \,+\, (I{\times}X);\ \mathsf{lunit}^{\cup}{+}\mathsf{proass}^{\cup} \ \circ \ \mathsf{rdist}^{\cup}]\!] \\
\in & [\mu(X \mapsto *R{+}(R{\times}X))] \cong [*R \times *R] \ .
\end{array}
$$

By transitivity it thus suffices to construct a witness to the simulation:

$$[*R] \gtrsim [\mu(X \mapsto *R{+}(R{\times}X))] \ .$$

This is achieved by applying the constructive Knaster-Tarski theorem.

$$
\begin{array}{ll}
 & \alpha \in [*R] \gtrsim [\mu(X \mapsto *R{+}(R{\times}X))] \\
\Leftarrow & \quad \{ \quad \text{constructive Knaster-Tarski: theorem 41} \\
 & \qquad \bullet \quad \alpha = (\![X \mapsto *I{+}(I{\times}X);\ \beta]\!) \ . \quad \} \\
 & \beta \in [*R] \gtrsim [*R \,+\, (R \times *R)] \\
\Leftarrow & \quad \{ \quad \text{disjoint sum: theorem 23}
\end{array}
$$

$$\bullet \quad \beta = \gamma \triangledown \delta \ . \ \}$$

$$\gamma \in [*R] \gtrsim [*R] \ \wedge \ \delta \in [*R] \gtrsim [R \times *R]$$

$$\Leftarrow \qquad \{ \quad \text{reflexivity, lists: (61)} \quad \}$$

$$\gamma = *I \ \wedge \ \delta = \mathsf{cons} \ .$$

Thus we have constructed

$$(\!|X \mapsto *I+(I{\times}X);\ *I \ \triangledown\ \mathsf{cons}|\!) \ \in\ [*R] \gtrsim [\mu(X \mapsto *R+(R{\times}X))] \ .$$

Composing the two witnesses together we obtain:

$$\begin{array}{ll} & (\!|X \mapsto *I+(I{\times}X);\ *I \ \triangledown\ \mathsf{cons}|\!) \\ \circ & [\!(X \ \mapsto\ *I + (I{\times}X);\ \mathsf{lunit}^{\cup}{+}\mathsf{proass}^{\cup} \ \circ\ \mathsf{rdist}^{\cup})\!] \\ \in & [*R] \cong [*R \times *R] \ . \end{array}$$

□

The combination illustrated above of a catamorphism after an anamorphism occurs very frequently, and has been dubbed a *hylomorphism* [21] . The anamorphism builds an element of the intermediate datatype $\mu(X \ \mapsto\ *I+(I{\times}X))$ which is then broken down by the catamorphism. A programmer would be more likely to have formulated list append as the least solution to the recursive equation

$$X :: \qquad X \ = \ *I \ \triangledown\ \mathsf{cons} \ \circ\ *I+(I{\times}X) \ \circ\ \mathsf{lunit}^{\cup}{+}\mathsf{proass}^{\cup} \ \circ\ \mathsf{rdist}^{\cup} \ .$$

In this form elements of the intermediate datatype are never explicitly constructed. For this reason it has been dubbed a *virtual* datatype [22].

The next example has been chosen for its relative difficulty, and for its practical relevance. An instance is the so-called “lines-unlines” problem [13]: given a sequence of two types of characters, delimiters and non-delimiters, write a program to divide the sequence into (possibly empty) “lines” of non-delimiters separated by single delimiters. Construct in addition the inverse of the program.

Expressed as an isomorphism between datatypes, the problem boils down to showing that the star decomposition theorem of regular langauges is constructively valid.

Theorem 64 (List Decomposition)

$$[*R \times *(S \times *R)] \cong [*(S{+}R)] \ .$$

Proof

$$\begin{array}{ll} & [*R \times *(S \times *R)] \\ = & \qquad \{ \quad \text{definition of } * \quad \} \\ & [*R \times \mu(X \mapsto 1\!\!1+((S \times *R){\times}X))] \\ \cong & \qquad \{ \quad \bullet \quad *I \times (\!|X \mapsto 1\!\!1+(I{\times}(*I \times X));\ 1\!\!1{+}\mathsf{proass}|\!) \ , \end{array}$$

$$
\begin{array}{ll}
 & \qquad\qquad \text{corollary 47} \quad \} \\
 & [*R \times \mu(X \mapsto 1\!\!1{+}(S{\times}(*R \times X)))] \\
= & \qquad \{ \qquad \text{rolling rule: (26)} \quad \} \\
 & [\mu(X \mapsto *R \times (1\!\!1{+}(S{\times}X)))] \\
\cong & \qquad \{ \qquad \bullet \quad (\![\ X \mapsto \mu(Y \mapsto (1\!\!1{+}(I{\times}X)){+}(I{\times}Y)) \\
 & \qquad\qquad\qquad ;\ (\![\ Y \mapsto (1\!\!1{+}(I{\times}I)){+}(I{\times}Y) \\
 & \qquad\qquad\qquad\quad ;\ \mathsf{rdist} \circ \mathsf{lunit}{+}\mathsf{proass}\ ]\!)\ ]\!), \\
 & \qquad\qquad \text{list fusion: (62) and corollary 47} \quad \} \\
 & [\mu(X \mapsto \mu(Y \mapsto (1\!\!1{+}(S{\times}X)){+}(R{\times}Y)))] \\
= & \qquad \{ \qquad \text{diagonal rule: (28)} \quad \} \\
 & [\mu(X \mapsto (1\!\!1{+}(S{\times}X)){+}(R{\times}X))] \\
\cong & \qquad \{ \qquad \bullet \quad (\![X \mapsto 1\!\!1{+}((I{+}I){\times}X);\ \mathsf{sumass} \circ 1\!\!1{+}\mathsf{rdist}^{\cup}]\!)\ , \\
 & \qquad\qquad \text{corollary 47} \quad \} \\
 & [\mu(X \mapsto 1\!\!1{+}((S{+}R){\times}X))] \\
= & \qquad \{ \qquad \text{definition of } * \quad \} \\
 & [*(S{+}R)] \ .
\end{array}
$$

We conclude that

$$
\begin{array}{ll}
 & *I \times (\![X \mapsto 1\!\!1{+}(I{\times}(*I \times X));\ 1\!\!1{+}\mathsf{proass}]\!) \\
\circ & (\![\ X \mapsto \mu(Y \mapsto (1\!\!1{+}(I{\times}X)){+}(I{\times}Y)) \\
 & ;\ (\![Y \mapsto (1\!\!1{+}(I{\times}I)){+}(I{\times}Y);\ \mathsf{rdist} \circ \mathsf{lunit}{+}\mathsf{proass}]\!)\ ]\!) \\
\circ & (\![X \mapsto 1\!\!1{+}((I{+}I){\times}X);\ \mathsf{sumass} \circ 1\!\!1{+}\mathsf{rdist}^{\cup}]\!) \\
\in & [*R \times *(S \times *R)] \cong [*(S{+}R)] \quad .
\end{array}
$$

□

The above, completely equational, calculation is a direct transcription of a proof in regular algebra of the identity, the only difference being the addition of witnesses at the three steps where only an isomorphism can be claimed. The witness is, however, rather complicated and is likely to lead to a rather inefficient implementation. This should not be seen, however, as a failing of the spec calculus. There are several ways to prove the decomposition law of which the above is just one, and a calculus that constrains one's freedom in constructing proofs is undesirable. It is however important that the calculus also embodies rules that allow one to transform programs to more efficient forms. The witness to the list decomposition theorem can be simplified considerably using the fixed point laws of the calculus. Use (37) to simplify the middle term in the witness. The fusion law (34) can then be used to fuse all three catamorphisms. In this way one obtains the witness

$$
\begin{array}{l}
(\![\ X \mapsto 1\!\!1{+}((I{+}I){\times}X) \\
;\ \mathsf{rdist} \circ \mathsf{lunit}{+}\mathsf{proass} \circ (1\!\!1{+}\mathsf{proass}){+}I \circ \mathsf{sumass} \circ 1\!\!1{+}\mathsf{rdist}^{\cup}\ ]\!)\ .
\end{array}
$$

The recursive equation corresponding to this catamorphism corresponds closely to the program that a programmer might construct using simple type arguments.

7 A Game of Leapfrog

The leapfrog rule in regular algebra takes the form

(65) $\quad x \times *(y{\times}x) \;=\; *(x{\times}y) \times x \quad .$

A related rule is

(66) $\quad x \times *x \;=\; *x \times x \quad .$

This latter rule is often cited as a special case of the leapfrog rule, obtained by instantiating y to $\mathbb{1}$. In this section we explore a general, constructive leapfrog rule of which the list leapfrog rule is a specific instance. The rule is obtained by replacing the operator $\times$ by a binary relator $\oslash$ and appropriately defining a unary relator to take the place of the $*$ operator. Roughly speaking the rule states that leapfrog properties are preserved by fixed point construction.

Somewhat surprisingly a generalisation of (66) can be established without recourse to a unit of multiplication. This is the first of the theorems below. Subsequently we prove the leapfrog preservation theorem and then apply it to the construction of a class of pairs of isomorphic relators of which the pair cons list, snoc list is an instance.

Theorem 67 Suppose F is a relator and $\oslash$ is a binary relator such that $(\oslash I)$ is universally $\sqcup$-junctive. Define the relator $\hat{F}$ by

$$\hat{F}.R = \mu(S \mapsto F.(R{\oslash}S)) \quad .$$

Suppose that

$$\mathsf{leapF} \in [R \oslash F.(S{\oslash}R)] \;\cong\; [F.(R{\oslash}S) \oslash R] \quad .$$

Then

$$[\![S \mapsto I \oslash F.S;\ \mathsf{leapF}]\!] \in [R \oslash \hat{F}.R] \;\cong\; [\hat{F}.R \oslash R] \quad .$$

Proof

$$\begin{array}{ll}
 & R \oslash \hat{F}.R \\
= & \qquad \{ \qquad \text{definition} \quad \} \\
 & R \oslash \mu(S \mapsto F.(R{\oslash}S)) \\
= & \qquad \{ \qquad \text{rolling rule: (26)} \quad \} \\
 & \mu(S \mapsto R \oslash F.S) \quad .
\end{array}$$

Thus,

$$
\begin{array}{ll}
& [\![S \mapsto I \oslash F.S;\ \mathsf{leapF}]\!] \ \in\ [R \oslash \hat{F}.R] \ \cong\ [\hat{F}.R \oslash R] \\
\equiv & \qquad \{ \qquad \text{above calculation, definition of } \hat{F} \quad \} \\
& \quad [\![S \ \mapsto\ I \oslash F.S;\ \mathsf{leapF}]\!] \\
& \in [\mu(S \mapsto R \oslash F.S)] \ \cong\ [\mu(S \mapsto F.(R \oslash S)) \oslash R] \\
\Leftarrow & \qquad \{ \qquad \text{(converse dual of) theorem 45 where} \\
& \qquad\qquad\quad R \oplus S \ = \ R \oslash F.S \quad \text{and} \quad R \otimes S = F.(R \oslash S) \quad \} \\
& \mathsf{leapF} \in [R \oslash F.(S \oslash R)] \ \cong\ [F.(R \oslash S) \oslash R] \ \ .
\end{array}
$$

□

Theorem 67 has an unbounded number of instances, some straightforward, others less obviously so. We content ourselves here with just one, the program implementing the "snoc" operation on "cons" lists.

Corollary 68

$$
[R \times *R] \cong [*R \times R] \ \ .
$$

Proof Since $*R = \mu(S \mapsto 1\!\!1 + (R \times S))$ we substitute $F := (1\!\!1+)$ and $\oslash := \times$ in the theorem. We obtain:

$$
\begin{array}{ll}
& [\![S \mapsto I \times (1\!\!1 + S);\ \gamma]\!] \ \in\ [R \times *R] \ \cong\ [*R \times R] \\
\Leftarrow & \gamma \ \in\ [R \times (1\!\!1 + (S \times R))] \ \cong\ [(1\!\!1 + (R \times S)) \times R] \ \ .
\end{array}
$$

Taking $\gamma \ = \ \mathsf{ldist} \ \circ \ (\mathsf{runit} \circ \mathsf{lunit}^{\cup}) + \mathsf{proass}^{\cup} \ \circ \ \mathsf{rdist}^{\cup}$ we have constructed the isomorphism. Specifically it is

$$
[\![S \mapsto I \times (1\!\!1 + S);\ \mathsf{ldist} \ \circ \ (\mathsf{runit} \circ \mathsf{lunit}^{\cup}) + \mathsf{proass}^{\cup} \ \circ \ \mathsf{rdist}^{\cup}]\!] \ \ .
$$

□

Theorem 69 (Leapfrog Preservation) Suppose F is a relator and $\oslash$ is a binary relator such that $(\oslash I)$ is universally $\sqcup$-junctive. Define the relator $\hat{F}$ by

$$
\hat{F}.R = \mu(S \mapsto F.(R \oslash S)) \ \ .
$$

Suppose that

$$
\mathsf{ass} \ \in\ [(R \oslash S) \oslash T] \ \cong\ [R \oslash (S \oslash T)]
$$

and that

$$
\mathsf{leapF} \in [R \oslash F.(S \oslash R)] \ \cong\ [F.(R \oslash S) \oslash R] \ .
$$

Then

$$
[R \oslash \hat{F}.(S \oslash R)] \cong [\hat{F}.(R \oslash S) \oslash R] \ \ .
$$

Proof We construct the witness as follows: First we have,

$$
\begin{array}{ll}
 & [R \oslash \hat{F}.(S{\oslash}R)] \\
= & \quad \{ \quad \text{definition of } \hat{F} \quad \} \\
 & [R \oslash \mu(X \mapsto F.((S{\oslash}R){\oslash}X))] \\
\cong & \quad \{ \quad \bullet \quad I{\oslash}(\!|X \mapsto F.(I{\oslash}(I{\oslash}X));\ F.\mathsf{ass}|\!)\ , \\
 & \qquad \text{corollary 47 and constructive monotonicity} \quad \} \\
 & [R \oslash \mu(X \mapsto F.(S{\oslash}(R{\oslash}X)))] \\
= & \quad \{ \quad \text{rolling rule: (26)} \quad \} \\
 & [\mu(X \mapsto R \oslash F.(S{\oslash}X))] \ .
\end{array}
$$

Thus

$$
\begin{array}{ll}
 & \alpha \ \in \ [R \oslash \hat{F}.(S{\oslash}R)] \ \cong \ [\hat{F}.(R{\oslash}S) \oslash R] \\
\Leftarrow & \quad \{ \quad \bullet \quad \alpha \ = \ I \oslash (\!|X \mapsto F.(I{\oslash}(I{\oslash}X));\ F.\mathsf{ass}|\!) \circ \beta \ , \\
 & \qquad \text{above calculation and constructive transitivity} \quad \} \\
 & \beta \ \in \ [\mu(X \mapsto R \oslash F.(S{\oslash}X))] \ \cong \ [\hat{F}.(R{\oslash}S) \oslash R] \\
\equiv & \quad \{ \quad \text{definition of } \hat{F} \quad \} \\
 & \beta \ \in \ [\mu(X \mapsto R \oslash F.(S{\oslash}X))] \ \cong \ [\mu(X \mapsto F.((R{\oslash}S){\oslash}X)) \oslash R] \\
\equiv & \quad \{ \quad \bullet \quad \beta = [\![X \ \mapsto \ I \oslash F.(I{\oslash}X);\ \gamma]\!] \\
 & \qquad \text{(converse dual of) constructive fusion: theorem 45} \\
 & \qquad \text{As in theorem 62 the relators } \oplus, \otimes \text{ and } \oslash \text{ in the theorem are} \\
 & \qquad \text{instantiated to } \textit{ternary} \text{ relators viewed as binary relators.} \\
 & \qquad \text{Specifically, we substitute} \\
 & \qquad (R{,}S){\oplus}X \ := \ F.((R{\oslash}S){\oslash}X) \ , \\
 & \qquad (R{,}S){\otimes}X \ := \ R \oslash F.(S{\oslash}X) \ , \text{ and} \\
 & \qquad X{\oslash}(R{,}S) \ := \ X{\oslash}R \ . \quad \} \\
 & \gamma \ \in \ [R \oslash F.(S{\oslash}(X{\oslash}R))] \ \cong \ [F.((R{\oslash}S){\oslash}X) \oslash R] \\
\Leftarrow & \quad \{ \quad \bullet \quad \gamma \ = \ I \oslash F.\mathsf{ass}^{\cup} \circ \delta \circ F.\mathsf{ass}^{\cup} \oslash I \\
 & \qquad \text{constructive monotonicity and transitivity} \quad \} \\
 & \delta \ \in \ [R \oslash F.(T{\oslash}R)] \ \cong \ [F.(R{\oslash}T) \oslash R] \\
\Leftarrow & \quad \{ \quad \text{assumption} \quad \} \\
 & \delta = \mathsf{leapF} \ .
\end{array}
$$

Summarising, the isomorphism we have constructed is

$$I{\oslash}(\!|X \mapsto F.(I{\oslash}(I{\oslash}X));\ F.\mathsf{ass}|\!)$$

$$\circ \quad [\![X \mapsto I \oslash F.(I \oslash X);\ I \oslash F.\mathsf{ass}^{\cup} \ \circ\ \mathsf{leapF} \ \circ\ F.\mathsf{ass}^{\cup} \oslash I]\!] \ .$$

Note that this is equal (using the rolling rule: (29)) to

$$(\![X \mapsto I \oslash F.(I \oslash X);\ I \oslash F.\mathsf{ass}]\!)$$

$$\circ \quad [\![X \mapsto I \oslash F.(I \oslash X);\ I \oslash F.\mathsf{ass}^{\cup} \ \circ\ \mathsf{leapF} \ \circ\ F.\mathsf{ass}^{\cup} \oslash I]\!]$$

which in turn is the least solution of the equation:

$$X :: \quad X \ = \ I \oslash F.(\mathsf{ass} \circ I \oslash X \circ \mathsf{ass}^{\cup}) \ \circ\ \mathsf{leapF} \ \circ\ F.\mathsf{ass}^{\cup} \oslash I \ .$$

It is (the equivalent of) this equation that a programmer might construct by type considerations.
□

Corollary 70 (List Leapfrog)

$$[R \times *(S \times R)] \cong [*(R \times S) \times R] \ .$$

Proof As in the proof of theorem 68 we make the substitutions

$$F \ := \ \mathbb{1}+$$

$$\oslash := \times$$

and

$$\mathsf{leapF} \ := \ \mathsf{ldist} \ \circ\ (\mathsf{runit} \circ \mathsf{lunit}^{\cup}) + \mathsf{proass}^{\cup} \ \circ\ \mathsf{rdist}^{\cup} \ .$$

To these must be added the substitution

$$\mathsf{ass} \ := \ \mathsf{proass} \ .$$

□

Theorems 67 and 69 can be dualised in the case that $(I\oslash)$ is universally $\sqcup$-junctive. The definition of the relator $\hat{F}$ must of course be dualised as well. More interesting is the case that both $(\oslash I)$ and $(I\oslash)$ are universally $\sqcup$-junctive because one then has two relators defined in terms of the given relator F, both of which enjoy the leapfrog property. What is the relationship between these two relators? Isomorphic, of course!

Theorem 71 Suppose F is a relator and $\oslash$ is a binary relator such that $(\oslash I)$ and $(I\oslash)$ are both universally $\sqcup$-junctive. Suppose that

$$\mathsf{leapF} \in [R \oslash F.(S \oslash R)] \ \cong \ [F.(R \oslash S) \oslash R] \ .$$

Define the relators $\hat{F}$ and $\check{F}$ by

$$\hat{F}.R = \mu(X \mapsto F.(R \oslash X)) \ ,$$

$$\check{F}.R = \mu(X \mapsto F.(X \oslash R)) \quad .$$

Then

$$\hat{F} \cong \check{F} \quad .$$

Proof On this occasion we are obliged to resort to a proof by mutual simulation. (In the near future we hope to have developed the theory of reductivity sufficiently to enable direct use of theorem 44 instead.)

First note that by theorem 67 we have an isomorphism $\hat{s}$ where

$$\hat{s} \in [R \oslash \hat{F}.R] \;\cong\; [\hat{F}.R \oslash R] \quad .$$

Specifically,

$$\hat{s} = [\![X \mapsto I \oslash F.X;\ \mathsf{leapF}]\!] \quad .$$

Dually (since $(I\oslash)$ is assumed to be universally $\sqcup$-junctive) we have, again by theorem 67 , an isomorphism $\check{s}$ where

$$\check{s} \in [\check{F}.R \oslash R] \;\cong\; [R \oslash \check{F}.R] \quad .$$

Specifically,

$$\check{s} = [\![X \mapsto F.X \oslash I;\ \mathsf{leapF}]\!] \quad .$$

Using these isomorphisms we construct simulations $\hat{t} \in \check{F} \gtrsim \hat{F}$ and $\check{t} \in \hat{F} \gtrsim \check{F}$ and then show that $\hat{t} = \check{t}^{\cup}$. First, the construction of $\hat{t}$.

$$
\begin{array}{ll}
 & \hat{t} \in \check{F} \gtrsim \hat{F} \\
\Leftarrow & \quad \{ \quad \hat{t} = (\![X \mapsto F.(I \oslash X);\ \alpha]\!) \ , \\
 & \quad\quad \text{constructive Knaster-Tarski: theorem 41} \quad \} \\
 & \alpha \in \check{F} \gtrsim [F.(R \oslash \check{F}.R)] \\
\Leftarrow & \quad \{ \quad \bullet \quad \alpha \ = \ \beta \circ F.\check{s} \ , \\
 & \quad\quad \text{constructive monotonicity and transitivity} \quad \} \\
 & \beta \in \check{F} \gtrsim [F.(\check{F}.R \oslash R)] \\
\Leftarrow & \quad \{ \quad \check{F}.R = F.(\check{F}.R \oslash R), \text{constructive reflexivity} \quad \} \\
 & \beta = \check{F}.I \quad .
\end{array}
$$

Thus, using trading to eliminate the domain restriction $\check{F}.I$, we obtain

$$(\![X \mapsto F.(I \oslash X);\ F.\check{s}]\!) \in \check{F} \gtrsim \hat{F} \quad .$$

Dually,

$$(\![X \mapsto F.(X \oslash I);\ F.\hat{s}]\!) \in \hat{F} \gtrsim \check{F} \quad .$$

We now show that these are inverses of each other. We have:

$$
\begin{array}{ll}
 & F.\hat{s} \\
= & \{ \quad \text{rolling rule: (29), definition of } \hat{s} \quad \} \\
 & [\![X \mapsto F.(I \oslash X);\ F.\mathsf{leapF}]\!] \quad .
\end{array}
$$

Similarly,

$$F.\check{s} \;=\; [\![X \mapsto F.(X \oslash I);\ F.\mathsf{leapF}]\!] \quad .$$

Thus,

$$
\begin{array}{ll}
 & (\![X \mapsto F.(X \oslash I);\ F.\hat{s}]\!)^{\cup} \;=\; (\![X \mapsto F.(I \oslash X);\ F.\check{s}]\!) \\
\equiv & \{ \quad \text{above calculation, } [\![T]\!] = (\![T^{\cup}]\!)^{\cup} \quad \} \\
 & \quad [\![X \mapsto F.(X \oslash I);\ (\![X \mapsto F.(I \oslash X);\ F.(\mathsf{leapF})^{\cup}]\!)]\!] \\
 & = (\![X \mapsto F.(I \oslash X);\ [\![X \mapsto F.(X \oslash I);\ F.(\mathsf{leapF})^{\cup}]\!]]\!) \\
\Leftarrow & \{ \quad \text{Fokkinga's theorem: [9]} \quad \} \\
 & F.(\mathsf{leapF})^{\cup} \;\in\; [F.(F.(I \oslash X) \oslash I)] \overset{\cdot}{\leftrightarrow} [F.(I \oslash F.(X \oslash I))] \\
\equiv & \{ \quad \text{assumption, naturality of } {}^{\cup} \text{ and relators} \quad \} \\
 & \mathit{true} \quad .
\end{array}
$$

□

Corollary 72 (Cons, Snoc List Isomorphism) Defining "cons" lists by the equation

$$\mathsf{Clist}.R \;=\; \mu(X \mapsto \mathbb{1} + (R \times X))$$

and "snoc" lists by the equation

$$\mathsf{Slist}.R \;=\; \mu(X \mapsto \mathbb{1} + (X \times R))$$

we have

$$\mathsf{Clist} \;\cong\; \mathit{Slist} \quad .$$

Proof Instantiate theorem 71 as in the proof of corollary 70 (omitting the assignment to ass which is unnecessary here).
□

8 Conclusion

By way of conclusion we would like to take this opportunity to expand on some of the remarks made in earlier sections and point out directions for future work.

The examples in this paper have been chosen to have some appeal to both theoreticians and practising programmers. In essence, we have shown that the theory of lists is "isomorphic" to constructive regular algebra in the same sense as the familiar "isomorphism" between propositions and types.

The examples have not been chosen idly. Elsewhere [7, 3, 11] we have argued the case for the central rôle of leapfrog and decomposition properties in algorithm development; the isomorphisms and simulations presented here are ones that, in our view, the ordinary programmer should be trained to recognise and exploit.

It is well-known that the equality of two regular expressions is decidable. Effectively we have shown here that all axioms of regular algebra are constructively true, with the exception of the axiom $x+x = x$. (This axiom cannot be constructively true; were it so, with isomorphism replacing equality, then we would have $\mathbb{1}+\mathbb{1} \cong \mathbb{1}$ which is clearly false since a two-element set cannot be placed in a bijection with a one-element set.) This raises the question of whether isomorphism between datatypes defined with the aid of the unit type, sum, product, and the list type is decidable. There is a strong possibility that the traditional algorithm for deciding equality of regular expressions can be adapted into an algorithm for constructing isomorphisms. It may even be the case that isomorphism tests on classes of "context-free" datatypes may be constructed by suitable adaptation of the known decision procedures for equality of context-free languages.

In the paper we have intentionally omitted discussion of the relationship between the properties of currying of functions and the properties of residuated lattices, in order to keep the length of the paper within bounds. The more general discussion of the relationship between Galois connections and adjunctions has also been omitted for the same reason. This is unfortunate from the point of view of completeness, but readers who have grasped the essential ideas presented here should be in a position to themselves make the necessary links.

The idea of extending proofs in lattice theory to constructions in category theory seems to be due to Lambek [19], as remarked in the introduction. That the process has a good chance of success is predicted, at least in part, by Reynolds' abstraction theorem [24]. The constructive Knaster-Tarski theorem, for example, expresses the "polymorphism" of catamorphism construction. (This observation was the inspiration for the development of the spec calculus. It has since become a standard example of the applications of the abstraction theorem.) The relationship of other theorems, for example the constructive fusion theorem, to the abstraction theorem is unclear, however, and the rôle of coherence properties does not seem to have been adequately explored. The clarification of which theorems of lattice theory can be made constructive, and why, is undoubtedly the most challenging question raised here.

Acknowledgement

Many thanks go to all those who have participated in the Dutch STOP (Specification and Transformation of Programs) project and to the members of the Mathematics of Program Construction Group for their enduring teamwork. Particular thanks go to Henk Doornbos for suggesting theorem 44 and to Harold Weffers for his help in producing the figures and in constructing the program texts.

This document was prepared using the Mathpad editing system. I am very grateful to its implementors Richard Verhoeven and Olaf Weber for creating such a pleasant interface.

References

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

[2] Jiří Adámek and Václav Koubek. Least fixed point of a functor. *J. of Comp. and Syst. Scs.*, 19:163–178, 1979.

[3] R.C. Backhouse. Making formality work for us. *EATCS Bulletin*, 38:219–249, June 1989.

[4] R.C. Backhouse. Monad decomposition. Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`, November 1992.

[5] 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 303–362. Springer-Verlag, Workshops in Computing, 1992.

[6] R.C. Backhouse, P. de Bruin, G. Malcolm, T.S. Voermans, and J. van der Woude. Relational catamorphisms. In Möller B., editor, *Proceedings of the IFIP TC2/WG2.1 Working Conference on Constructing Programs*, pages 287–318. Elsevier Science Publishers B.V., 1991.

[7] R.C. Backhouse and B.A. Carré. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and its Applications*, 15:161–186, 1975.

[8] R.C. Backhouse and H. Doornbos. Induction and recursion on datatypes. Department of Computing Science, Eindhoven University of Technology. Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`, 1993.

[9] R.C. Backhouse, H. Doornbos, and P. Hoogendijk. Commuting relators. Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`, September 1992.

[10] R.C. Backhouse and P. Hoogendijk. Elements of a relational theory of datatypes. To appear. Presented at IFIP Working Group 2.1 state of the art summer school, Itacuruçá Island, Brazil, January 10-23, 1992.

[11] Roland Backhouse and A.J.M. van Gasteren. Calculating a path algorithm. In R.S. Bird, C.C. Morgan, and J.C.P. Woodcock, editors, *Mathematics of Program Construction. 2nd International Conference, June/July 1992*, volume 669 of *Lecture Notes in Computer Science*, pages 32–44. Springer Verlag, 1993.

[12] M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer-Verlag, 1985.

[13] R.S. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice-Hall, 1988.

[14] J.H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, London, 1971.

[15] P.J. Freyd and A. Scedrov. *Categories, Allegories.* North-Holland, 1990.

[16] P.F. Hoogendijk. (Relational) Programming laws in the Boom hierarchy of types. In R.S. Bird, C.C. Morgan, and J.C.P. Woodcock, editors, *Mathematics of Program Construction. 2nd International Conference, June/July 1992*, volume 669 of *Lecture Notes in Computer Science*, pages 163–190. Springer Verlag, 1993. Extended version to appear in Science of Computer Programming.

[17] G. Hutton and E. Voermans. Making functionality more general. In *Functional Programming, Glasgow 1991*, Workshops in computing. Springer-Verlag, 1991.

[18] S.C. Kleene. Representation of events in nerve nets and finite automata. In Shannon and McCarthy, editors, *Automata Studies*, pages 3–41. Princeton Univ. Press, 1956.

[19] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[20] S. MacLane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971.

[21] E. Meijer, M.M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *FPCA91: Functional Programming Languages and Computer Architecture*, volume 523 of *LNCS*, pages 124–144. Springer-Verlag, 1991.

[22] de O. Moor and D.S. Swierstra. Virtual data structures. To appear. Presented at IFIP Working Group 2.1 state of the art summer school, Itacuruçá Island, Brazil, January 10-23, 1992.

[23] Vaughan Pratt. Action logic and pure induction. In J. van Eijck, editor, *Proc. Logics in AI: European Workshop JELIA '90*, volume 478, pages 97–120. Springer-Verlag Lecture Notes in Computer Science, 1990.

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

[25] J. Riguet. Relations binaires, fermetures, correspondances de Galois. *Bulletin de la Société Mathématique de France*, 76:114–155, 1948.

[26] D.E. Rydeheard and R.M. Burstall. *Computational Category Theory.* Prentice-Hall International, 1988.

[27] Dana Scott. Data types as lattices. *SIAM J. of Computing*, 5(3):522–587, September 1976.

[28] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761–83, 1982.

[29] E. Voermans. Pers as types, inductive types and types with laws. In *PHOENIX Seminar and Workshop on Declarative Programming, Sasbachwalden*, Workshops in Computing. Springer-Verlag, 1991.

[30] Ed Voermans and Jaap van der Woude. A relational perspective on types with laws. Presented at the informal workshop on Categories of Relations in Computer Science, Oxford, July 1993.

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

[32] P. Wadler. Church and state: taming effects in functional languages. Marktoberdorf Summer School, 1992.