Categorical models of reversible computing

Robin Kaarsgaard[1] and Mathys Rennela[2]

1 **University of Copenhagen**
Copenhagen, Denmark
`robin@di.ku.dk`

2 **Radboud University**
Nijmegen, The Netherlands
`mathysr@cs.ru.nl`

Abstract

Reversible computing is a computational paradigm in which computations are deterministic in both the forward and backward direction, such that programs have well-defined forward *and* backward semantics. In the present paper, we investigate the formal semantics of the reversible functional programming language Rfun. We focus on inverse categories, which are categories with an abstract notion of partiality and whose morphisms have partial inverses. We determine in which inverse categories one can define a categorical object in which every term of the language Rfun has a denotation. We axiomatize a class of categorical models of reversible computing, which provide computationally adequate denotational semantics for Rfun, confirming the importance of inverse category theory in the study of reversible computation.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases domain theory, reversible computing, inverse categories, categorical semantics

1 Introduction

Since the early days of theoretical computer science, the quest for the mathematical description of (functional) programming languages has led to a substantial body of work. In particular, much focus has been put on computationally adequate models, which equip programs with a sound and complete denotational semantics.

In reversible computing, every program is *reversible*, i.e., both *forward* and *backward* deterministic. But why study such a peculiar paradigm of computation at all?

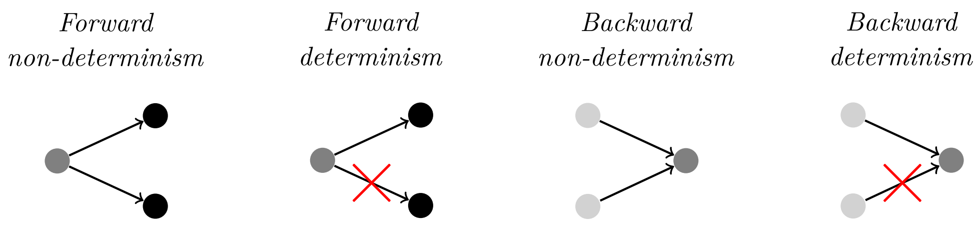
While the daily operations of our computers are irreversible, the physical devices which execute them are fundamentally reversible. In the paradigm of quantum computation, the physical operations performed by a scalable quantum computer intrinsically rely on quantum mechanics, which is reversible. Landauer [22] has famously argued, through what has later been coined *Landauer's principle*, that the erasure of a bit of information is inexorably linked to the dissipation of energy as heat (which has since seen both formal [1] and experimental [8] verification). On its own, this constitutes a reasonable argument for the study of reversible computing, as this model of computation sidesteps this otherwise inevitable energy dissipation by avoiding the erasure of information altogether.

And although its study can be motivated by issues raised by the laws of thermodynamics which arguably constitute a theoretical limit of Moore's law [26], reversibility arises not only in quantum computing (see e.g., [3]), but also has its own circuit models [13, 32], Turing machines [4, 7] and other automata [20, 21], and has seen applications in areas spanning from high-performance computing [29] to process calculi [12] and robotics [30, 31], to name a few.

Even if mainstream theoretical computer science has always put the focus on irreversible computations, there is an increasing interest for the study of the reversible computing paradigm through the lens of the theoretical computer scientist's toolbox. The present work provides a building block in the theoretical study of reversible computing, studied with the mathematical methods provided by the theory of the semantics of programming languages: we describe computationally adequate categorical models of reversible computing.

1.1 Reversible programming primer

Virtually all programming languages in use today (with some notable exceptions) guarantee that programs are *forward deterministic* (or simply *deterministic*), in the sense that any current computation state uniquely determines the next computation state. Few such languages, however, guarantee that programs are *backward deterministic*, *i.e.*, that any current computation state uniquely determines the *previous* computation state. For example, assigning a constant value to a variable in an imperative programming language is forward deterministic, but not backward deterministic (as one generally has no way of determining the value stored in the variable prior to this assignment).



Programming languages which guarantee both forward and backward determinism of programs are called *reversible*. Though there are numerous examples of such reversible programming languages (see, *e.g.*, [17, 34]), we focus here on the reversible functional programming language Rfun [33].

Rfun is an untyped reversible functional programming language (see an example program, computing Fibonacci pairs, to the right, due to [33]) similar in style to the Lisp family of programming languages. As a consequence of reversibility, all functions in Rfun are *partial injections*, *i.e.*, whenever some function f is defined at points x and y, $f(x) = f(y)$ implies $x = y$. Being untyped, values in Rfun come in the form of Lisp-style symbols and constructors.

$$\begin{array}{lll}
\mathit{plus}\ \langle x,y\rangle \triangleq \textbf{case}\ y\ \textbf{of} & & \\
\qquad Z & \rightarrow & \lfloor\langle x\rangle\rfloor \\
\qquad S(u) & \rightarrow & \textbf{let}\ \langle x',u'\rangle = \mathit{plus}\ \langle x,u\rangle\ \textbf{in} \\
 & & \langle x', S(u')\rangle \\
 & & \\
\mathit{fib}\ n \triangleq \textbf{case}\ n\ \textbf{of} & & \\
\qquad Z & \rightarrow & \langle S(Z), S(Z)\rangle \\
\qquad S(m) & \rightarrow & \textbf{let}\ \langle x,y\rangle = \mathit{fib}\ m\ \textbf{in} \\
 & & \textbf{let}\ z = \mathit{plus}\ \langle y,x\rangle\ \textbf{in}\ z
\end{array}$$

Pattern matching and variable binding is supported by means of (slightly restricted) forms of case expressions and let bindings, and iteration by means of general recursion. Restrictions on case expressions are there to ensure that the different branches (seen as morphisms) are inverse compatible. That is, the syntax of Rfun ensures that the inverse of a case expression is also a case expression.

In order to guarantee backward determinism (and consequently reversibility), Rfun imposes a few restrictions on function definitions not usually present in irreversible functional programming languages. Firstly, while a given variable may only appear once in a pattern (as is also the case in, *e.g.*, Haskell and the ML family), it must also occur exactly once in the body.

```
plus^{-1} z ≜ case z of
        ⌊⟨x⟩⌋        →  ⟨x, Z⟩
        ⟨x', S(u')⟩  →  let ⟨x, u⟩ = plus^{-1} ⟨x', u'⟩ in
                        ⟨x, S(u)⟩

 fib^{-1} z ≜ case z of
        ⟨S(Z), S(Z)⟩ →  Z
        z'           →  let ⟨y, x⟩ = plus^{-1} z' in
                        let m = fib^{-1} ⟨x, y⟩ in S(m)
```

Secondly, the result of a function call must be bound in a let binder before use; and thirdly, the leaf expression of any branch in a case expression must not match *any* leaf expression in a branch preceding it. Together, these three restrictions guarantee reversibility. One might wonder whether these hinder expressivity too much; fortunately, this is not so, as Rfun is r-Turing complete [33], that is: Rfun can simulate any reversible Turing machine [7].

A final peculiarity of Rfun has to do with duplication of values. Even though values *can* be (de)duplicated reversibly, the linear use policy on variables hinders this. To allow for (de)duplication of values, a special *duplication/equality* operator $\lfloor \cdot \rfloor$ is introduced. Note that this operator can be used both as a value and as a pattern; in the former use case, it is used to (de)duplicate values, and in the latter, to test whether values are identical.

$$\lfloor \langle x \rangle \rfloor = \langle x, x \rangle$$

$$\lfloor \langle x, y \rangle \rfloor = \begin{cases} \langle x \rangle & \text{if } x = y \\ \langle x, y \rangle & \text{if } x \neq y \end{cases}$$

As a reversible programming language, Rfun has the property that it is syntactically closed under inverses. That is to say, if p is an Rfun program, there exists another Rfun program p' such that p' computes the semantic inverse of p. This is witnessed by a *program inverter* [33]. For example, the inverses to addition and Fibonacci pair functions shown earlier is given above.

1.2 Join inverse category theory

In Section 2, we focus on categorical structures in which one can conveniently model the mathematical foundations of reversible computing. A brainchild of Cockett & Lack [9–11], restriction categories are categories with an abstract notion of partiality, associating to each morphism $f : A \to B$ a "partial identity" $\overline{f} : A \to A$ satisfying $f \circ \overline{f} = f$ and other axioms. In particular, this gives rise to *partial isomorphisms*, which are morphisms $f : A \to B$ associated with partial inverses $f^\dagger : B \to A$ such that $f^\dagger \circ f = \overline{f}$ and $f \circ f^\dagger = \overline{f^\dagger}$. In this context, a *total* map is a morphism $f : A \to B$ such that $\overline{f}$ is the identity on A.

A restriction category in which all morphisms are partial isomorphisms is called an *inverse category*. Such categories have been considered by the semigroup community for decades [16, 19, 23], though they have more recently been rediscovered in the framework of restriction category theory, and considered as models of reversible computing [5, 14, 15, 18]. The category **PInj** of sets and partial injections is the canonical example of an inverse category. In fact, every (locally small) inverse category can be faithfully embedded in it [9].

In line with [5], we focus here on a particular class of domain-theoretic inverse categories, namely join inverse categories. Informally, join inverse categories are inverse categories in which joins (i.e. least upper bounds) of homomorphisms exist in such a way that the partial identity of a join is the join of the partial identities, among with other coherence axioms.

In this setting, bimonoidal join inverse categories are join inverse categories equipped with products $X \otimes Y$ and sums $X \oplus Y$ for every pair of objects X and Y, together with a natural isomorphism which distributes products over sums, and coherence axioms which ensure the preservation of joins and partial inverses. Interestingly, every inverse category embeds in such a bimonoidal join inverse category (see Theorem 13).

1.3 Categorical models of reversible computing

The major part of this work is devoted to the description of an axiomatization of categorical models of reversible computing, following a recent keen interest for presheaf-theoretic models in the semantics of quantum computing [24, 27, 28] and reversible computing [5].

In order to construct denotational models of the terms of the language Rfun, we adopt the categorical formalism of bimonoidal join inverse categories. Since morphisms in inverse categories are all partial isomorphisms, inverse categories have been suggested as models of reversible functional programming languages [14], and the presence of joins has been shown [5, 18] to induce fixed point operators for modelling reversible recursion.

In short, Rfun is an untyped first-order language in which the arguments of functions are organized in left expressions (patterns) given by the grammar $l ::= x \mid c(l_1, \dots, l_n)$ where variables x and constructors c are taken from denumerable alphabets $\mathcal{V}$ respectively Σ. In other words, a function definition $f\, l \triangleq l'$ takes a pattern l as argument and organizes its output as a pattern l'.

Our work involves an unorthodox way of thinking about the denotational semantics of a function in a functional programming language to be explicitly constructed piecemeal by the (partial) denotations of its individual branches. Categorically, this means that the denumerable alphabet Σ is denoted by the (least) fix point of the functor $F : X \mapsto X \oplus 1$, which is given by algebraic compactness as the initial F-algebra [6] and corresponds to the denotation of the recursive type of natural numbers. Then, every value $c(l_1, \dots, l_n)$ is naturally denoted by induction as a tree which has a root labelled by the symbol c, with a branch to every subtree l_i for $1 \leqslant i \leqslant n$. We detail this construction in Section 3.

Finally, in order to denote the duplication/equality operator and the case expressions of Rfun, we investigate the notions of *decidable equality* and *decidable pattern matching.* Recall that in set theory, a set is *decidable* (or *has decidable equality*) whenever any pair of elements is either equal or different. By decidable equality, we mean that the equality of two elements is decidable from a computability-theoretic point of view[1]. Similarly, decidable pattern matching is the property which ensures that checking for some pattern in a given sequence of indexes is a decidable computation. Those are two properties that we want to hold, regardless of the terms that we are dealing with.

Section 4 is devoted to the study of decidability in join inverse categories. Interestingly, it is sufficient to assume that the object 1 has decidable equality to obtain decidable equality for all objects relevant to the denotational semantics (see Section 4.2). Moreover in join inverse category theory, decidable equality implies decidable pattern matching (see Section 4.3).

We conclude in Section 5 with a definition of categorical models of reversible computing as bimonoidal join inverse categories with decidable equality. Then, in any categorical model of reversible computing $\mathbf{C}$, Rfun expressions are denoted by elements of objects of $\mathbf{C}$ and function definitions are denoted by maps in $\mathbf{C}$, providing a denotational semantics for the language Rfun [33] whose operational semantics can be associated to the following adequacy theorem:

[1] Note that we're only talking about decidable equality of inductively constructed first order data, that is data constructed using symbols and constructors. A more general notion of decidable equality would be at odds with Turing completeness.

▸ **Theorem 1.** *In any categorical model of reversible computing and under any programming context, a program execution terminates if and only if its denotation is a non-trivial total map.*

This axiomatization gives for the first time (to our knowledge) reasonable axioms for categorical semantics of reversible computing and paves the way to the quest for fully abstract models of reversible computing, as much as it gives to the working computer scientist some guidelines for the design of reversible functional programming languages and their compilers.

2 Join inverse category theory

Assuming prior knowledge of category theory, we introduce in this section the notion of join inverse category, due to Cockett & Lack [9].

▸ **Definition 2.** A *restriction structure* on a category consists of an operator mapping each morphism $f : A \to B$ to a morphism $\overline{f} : A \to A$ (called *restriction idempotent*) such that the following properties are satisfied:

(i) $f \circ \overline{f} = f$,
(ii) $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$ for all $g : A \to C$,
(iii) $\overline{g \circ \overline{f}} = \overline{g} \circ \overline{f}$ for all $g : A \to C$,
(iv) $\overline{g} \circ f = f \circ \overline{g \circ f}$ for all $g : B \to C$.

A category with a restriction structure is called a *restriction category*. A *total* map is a morphism $f : A \to B$ such that $\overline{f} = \mathrm{id}_A$.

We recall some basic properties of restriction idempotents:

▸ **Lemma 3.** *In any restriction category, we have for all suitable f and g that*

(i) $\overline{f} \circ \overline{f} = \overline{f}$,
(ii) $\overline{g \circ f} = \overline{\overline{g} \circ f}$,
(iii) $\overline{\overline{g} \circ \overline{f}} = \overline{g} \circ \overline{f}$, *and*
(iv) $\overline{g \circ f} \circ \overline{f} = \overline{g \circ f}$.

As a trivial example, any category is a restriction category when equipped with the trivial restriction structure mapping $\overline{f} = 1_A$ for all $f : A \to B$.

▸ **Definition 4.** A morphism $f : A \to B$ in a restriction category is a partial isomorphism whenever there exists a morphism $f^\dagger : B \to A$, the partial inverse of f, such that $f^\dagger \circ f = \overline{f}$ and $f \circ f^\dagger = \overline{f^\dagger}$.

Note that the definite article – *the* partial inverse – is justified, as partial inverses are unique whenever they exist.

▸ **Definition 5.** An *inverse category* is a restriction category in which every morphism is a partial isomorphism.

It is worth noting that this is not the only definition of an inverse category: historically, this mathematical structure has been defined as the categorical extension of inverse semigroups rather than as a particular class of restriction categories (see, *e.g.*, [16, 19, 23]).

▸ **Definition 6.** A zero object in a restriction (or inverse) category is said to be a *restriction zero* iff $\overline{0_{A,A}} = 0_{A,A}$ for every zero endomorphism $0_{A,A}$.

▸ **Definition 7.** Parallel morphisms $f, g : A \to B$ of an inverse category are said to be *inverse compatible*, denoted $f \asymp g$, if the following hold:

(i) $g \circ \overline{f} = f \circ \overline{g}$ (ii) $g^\dagger \circ \overline{f^\dagger} = f^\dagger \circ \overline{g^\dagger}$

By extension, one says that $S \subseteq \mathrm{Hom}(A, B)$ is inverse compatible if $s \asymp t$ for each $s, t \in S$.

More specifically, we focus in this paper on join inverse categories. The definition of such categories relies on the fact that in a restriction category $\mathbf{C}$, every hom-set $\mathbf{C}(A, B)$ gives rise to a poset when equipped with the following partial order: $f \leqslant g$ if and only if $g \circ \overline{f} = f$.

▸ **Definition 8** ([15]). An inverse category is a (countable) *join inverse category* if it has a restriction zero object, and satisfies that for all (countable) inverse compatible subsets S of all hom sets $\mathrm{Hom}(A, B)$, there exists a morphism $\bigvee_{s \in S} s$ such that

(i) $s \leqslant \bigvee_{s\in S} s$ for all $s \in S$, and $s \leqslant t$ for all $s \in S$ implies $\bigvee_{s\in S} s \leqslant t$;
(ii) $\overline{\bigvee_{s\in S} s} = \bigvee_{s\in S} \overline{s}$;
(iii) $f \circ (\bigvee_{s\in S} s) = \bigvee_{s\in S}(f \circ s)$ for all $f : B \to X$; and
(iv) $(\bigvee_{s\in S} s) \circ g = \bigvee_{s\in S}(s \circ g)$ for all $g : Y \to A$.

On that matter, it is important to mention that there are significant mathematical results about join *inverse* categories. In particular, there is an adjunction between the categories of join restriction categories and join inverse categories [5].

Let us conclude the introduction of this section with a few examples.

The category **PInj** of sets and partial injections is a canonical example of an inverse category (even further, by the categorical Wagner-Preston theorem [9], every (locally small) inverse category can be faithfully embedded in **PInj**). For a partial injection $f : A \to B$, define its restriction idempotent $\overline{f} : A \to A$ by $\overline{f}(x) = x$ if f is defined at x, and undefined otherwise. With this definition, every partial injection is a partial isomorphism. Moreover, the partial order on homsets corresponds to the usual partial order on partial functions: that is, for $f, g \in \mathbf{PInj}(A, B)$, $f \leqslant g$ if and only if, for every $x \in A$, f is defined at x implies that g is defined at x in such a way that $f(x) = g(x)$.

The category $\mathbf{Set}_*$ of pointed sets and point-preserving functions has an analogous restriction structure. Other examples of join inverse categories include: The category **PTop** of topological spaces and partial homeomorphisms with open range and domain of definition. Restrictions are given as in $\mathbf{Set}_*$, and the existence of joins follows by the same construction as in $\mathbf{Set}_*$ (openness follows by the pasting lemma). Relatedly, the category $\mathbf{Dcpo}_{\perp !}$ of directed complete partial orders and partial Scott homeomorphisms with open range and domain of definition (recall that the Scott topology is generated by all downsets $\downarrow\{x\} = \{y \mid y \leqslant x\}$) is a join inverse category.

2.1 Bimonoidal join inverse categories

We now proceed to introduce our categorical model: bimonoidal join inverse categories.

▸ **Definition 9** ([5, 14]). A join inverse category $\mathbf{C}$ with a restriction zero object 0 is said to have a *join inverse sum* if it is equipped with a symmetric monoidal join restriction functor $- \oplus - : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ (with left unitor $\lambda_\oplus$, right unitor $\rho_\oplus$, associator $\alpha_\oplus$, and commutator $\gamma_\oplus$) such that the restriction zero 0 is the tensor unit, and the morphisms given by $\amalg_1 = (1_A \oplus 0_{0,B}) \circ \rho^{-1} : A \to A \oplus B$ and $\amalg_2 = (0_{0,B} \oplus 1_A) \circ \lambda^{-1} : A \to B \oplus A$ are jointly epic, and their partial inverses $\amalg_1^\dagger : A \oplus B \to A$ and $\amalg_2^\dagger : B \oplus A \to A$ are jointly monic.

This notion is intimately related to B. Giles' notion of a *disjointness tensor* – it is simply such a tensor required to preserve joins, so precisely what is described as a join-preserving disjointness tensor in [5].

At this point, to define the notion of an inverse product (which first appeared in [14]), we recall the definition of a †-Frobenius semialgebra (see, *e.g.*, [14]), that we later use to describe well-behaved products on inverse categories.

▸ **Definition 10.** In a (symmetric) monoidal †-category, a *†-Frobenius semialgebra* is a pair (X, μ_X) of an object X and a map $\mu_X : X \otimes X \to X$ such that the diagrams below commute.

$$\begin{tikzcd}
X \otimes (X \otimes X) \arrow[rr, "\alpha"] \arrow[d, "X \otimes \mu_X"'] & & (X \otimes X) \otimes X \arrow[d, "\mu_X \otimes X"] \\
X \otimes X \arrow[dr, "\mu_X"'] & & X \otimes X \arrow[dl, "\mu_X"] \\
& X &
\end{tikzcd}
\qquad
\begin{tikzcd}
X \otimes X \arrow[rr, "\alpha \circ (\mu_X^\dagger \otimes X)"] \arrow[dd, "\alpha^{-1} \circ (X \otimes \mu_X^\dagger)"'] \arrow[dr, "\mu_X"] & & X \otimes (X \otimes X) \arrow[dd, "X \otimes \mu_X"] \\
& X \arrow[dr, "\mu_X^\dagger"] & \\
(X \otimes X) \otimes X \arrow[rr, "\mu_X \otimes X"'] & & X \otimes X
\end{tikzcd}$$

Formally, the leftmost diagram (and its dual) makes (X, μ_X) a semigroup, and$(X, \mu_X^\dagger)$ a cosemigroup, while the diagram to the right is called the *Frobenius condition*. One says that a †-Frobenius semialgebra (X, μ_X) is *special* if $\mu_X \circ \mu_X^\dagger = \mathrm{id}_X$, *cospecial* if $\mu_X^\dagger \circ \mu_X = \mathrm{id}_{X \otimes X}$, and *commutative* if the monoidal category in which it lives is symmetric and $\mu_X \circ \gamma_{X,X} = \mu_X$ (where $\gamma_{X,X} : X \otimes X \to X \otimes X$ is the symmetry of the monoidal category).

Next, we recall the definition of a (join) inverse product.

▸ **Definition 11.** An inverse category $\mathbf{C}$ is said to have an *inverse product* [14] if it is equipped with a symmetric monoidal restriction functor $- \otimes - : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ (with left unitor $\lambda_\otimes$, right unitor $\rho_\otimes$, associator $\alpha_\otimes$, and commutator $\gamma_\otimes$) equipped with a natural transformation $\Delta_X : X \to X \otimes X$ such that the pair (X, Δ_X) is a special and commutative †-Frobenius semialgebra for any object X. Additionally, one says that a join inverse category has a *join inverse product* if it has an inverse product that additionally preserves joins.

When clear from the context, we omit the subscripts on unitors, associators, and commutators. We are finally ready to define bimonoidal join inverse categories.

▸ **Definition 12.** A join inverse category is *bimonoidal* if it is equipped with an join inverse product $(\otimes, 1)$ and a join inverse sum $(\oplus, 0)$, such that there is a natural isomorphism of join restriction functors $X \otimes (Y \oplus Z) \xrightarrow{\delta} (X \otimes Y) \oplus (X \otimes Z)$ (the *distributor*) natural in X, Y, and Z, and a natural isomorphism $X \otimes 0 \xrightarrow{\nu_X} 0$ (the *annihilator*) natural in X.

The presence of the annihilator is of particular importance in relation to Call-by-Value semantics, as it "eagerly evaluates" the partiality of monoidal product maps (similar to the smash product of topological spaces, dcpos, etc.), in the sense that tensoring any $f : A \to C$ with a zero map, *i.e.*, $f \otimes 0_{B,D} : A \otimes B \to C \otimes D$ (or symmetrically), gives the zero map $0_{A \otimes B, C \otimes D}$ by commutativity of the diagram below, and unicity (up to canonical isomorphism) of the zero object.

$$\begin{tikzcd}
A \otimes B \arrow[rr, "f \otimes 0_{B,D}"] \arrow[dr, "A \otimes !"'] & & C \otimes D \\
& A \otimes 0 \arrow[ur, "f \otimes !"'] \arrow[d, bend right, "\nu_A"] & \\
& 0 \arrow[u, bend right, "\nu_A^{-1}"] &
\end{tikzcd}$$

A particularly useful result in this regard is the following.

▸ **Theorem 13.** *Any inverse category can be faithfully embedded in a category of which is*

(i) a bimonoidal join inverse category,
(ii) algebraically ω-compact for join restriction functors.

Proof. See Appendix A.1. ◂

The proof of this theorem uses a modified version of Guo's completeness theorem for join inverse categories [15] (see also [5]). In light of this, we assume our bimonoidal join inverse categories from here on out to be algebraically ω-compact for join restriction endofunctors: every join restriction endofunctor F has a fixpoint written μF (or alternatively $\mu X.F(X)$), which is the initial F-algebra.

On a final note, it is worthwhile to stress that our definition of bimonoidality is fairly similar to the notion of distributive join inverse category in Giles' terminology [14, Sec. 9.2]: there, the case of general joins is not considered and therefore there is no requirement of join-preservation; however, for reasons unrelated to our approach, the existence of certain pullbacks and pushouts of his related disjointness tensor are required.

3 Constructions on bimonoidal join inverse categories

In this section, we construct the object $T\mathcal{S}$ of all Rfun terms, allowing us to construct the denotation $\text{lift}(s) \in T\mathcal{S}$ for every variable x such that $\sigma(x) = s$, and the denotations $\text{cons}_c(s_1, \dots, s_n) \in T\mathcal{S}$ of left expressions $c(l_1, \dots, l_n)$ where $\sigma = \uplus_{i=1}^{n} \sigma_i$ and s_i is the denotation of l_i (for each i). We refer the interested reader to Appendix B for the complete presentation of our computationally adequate denotational semantics of Rfun. There, the denotation $[\![t]\!]_q^\sigma$ of a term t depends on the program q associated to the term t and on the substitution σ, a partial function which associates variables to symbols taken in a denumerable alphabet.

3.1 Term representation

Now, we need to deal with the fact that values in the untyped functional programming language Rfun are left expressions free of variables and duplication/equality [33], *i.e.* of the form $v := s \mid c(v_1, \dots, v_n)$ where s and c are taken from a denumerable alphabet Σ.

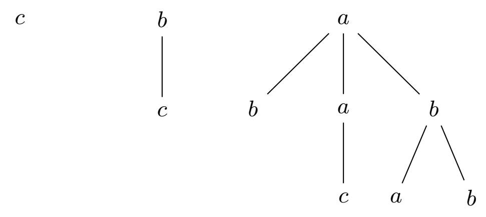
Then, to obtain an object of Rfun terms, we need to construct a denumerable object $\mathcal{S}$ of *symbols* or *base terms*, each identified by a unique morphism $1 \to \mathcal{S}$ where 1 is unit of the inverse product – a monoidal natural numbers object suffices (up to isomorphism). We define $\mathcal{S}$ to be the least fixed point of the locally continuous functor $X \mapsto X \oplus 1$, via algebraic compactness: In the following *non-commutative* diagram, every map $\mu_n \circ \amalg_{n+1}$ gives a unique symbol of $\mathcal{S}$.

$$\begin{tikzcd}
 & 1 \arrow[dl, "\text{id}"'] \arrow[d, "\amalg_2"'] \arrow[dr, "\amalg_3"] \arrow[drr, "\cdots"] & & \\
1 \arrow[dr, "\mu_0"'] & 1 \oplus 1 \arrow[d, "\mu_1"'] & 1 \oplus 1 \oplus 1 \arrow[dl, "\mu_2"'] & \cdots \arrow[dll, "\cdots"] \\
 & \mathcal{S} & &
\end{tikzcd}$$

Building up on Theorem 13 and using the join inverse product $(\otimes, 1)$ and join inverse sum $(\oplus, 0)$, we can construct the object of Rfun terms as $T\mathcal{S}$, where the functor T is defined as follows:

$$T(X) = \mu K.X \otimes L(K) \qquad L(X) = \mu K.1 \oplus (X \otimes K)$$

Intuitively, L maps an object X to that of lists of X, while T maps an object X to nonempty finite trees with X-values at each node. In **Set**, the following are examples of elements of $T(X)$ (for some set X and $a, b, c \in X$):



(1) (2) (3)

Note that while every such tree is well-founded, the converse is not true, as well-founded trees can be empty or have infinite breadth, neither of which are permitted by this definition.

The object $T\mathcal{S}$ then allows us to represent terms in Rfun very naturally, namely by their syntax trees. As such, (1) above corresponds to the term c, (2) to the term $b(c)$, and (3) to the term $a(b, a(c), b(a, b))$. Though this is often how untyped programming languages are modelled, we do not formally require $T\mathcal{S}$ to be an universal object of the category, as long as it is rich enough for $\mathbf{C}(1, T\mathcal{S})$ to uniquely encode all values.

3.2 Tags

To construct new terms from simpler ones, and to decompose complex terms into its constituent parts, we introduce a family of *tagging* and *untagging* maps indexed by symbols, *i.e.*, morphisms $1 \to \mathcal{S}$.

Let us start by defining maps $\text{tag}_s : LT\mathcal{S} \to T\mathcal{S}$ and $\text{untag}_s : T\mathcal{S} \to LT\mathcal{S}$ for every symbol $s : 1 \to \mathcal{S}$. The second map happens to coincide precisely with the partial inverse of the corresponding tagging map. One can think of the partial inverse of a symbol $s : 1 \to \mathcal{S}$ as its *assertion* $\text{asrt}_s : \mathcal{S} \to 1$. intuitively, it is a partial map defined at a unique point corresponding to the symbol it asserts.

$$\begin{tikzcd}
LT\mathcal{S} \arrow[r, "\text{tag}_s"] \arrow[d, "\lambda^{-1}"'] & T\mathcal{S} \\
1 \otimes LT\mathcal{S} \arrow[r, "s \otimes \text{id}"'] & \mathcal{S} \otimes LT\mathcal{S} \arrow[u, "\text{fold}_T"']
\end{tikzcd}
\qquad
\begin{tikzcd}
T\mathcal{S} \arrow[r, "\text{untag}_s"] \arrow[d, "\text{unfold}_T"'] & LT\mathcal{S} \\
\mathcal{S} \otimes LT\mathcal{S} \arrow[r, "\text{asrt}_s \otimes \text{id}"'] & 1 \otimes LT\mathcal{S} \arrow[u, "\lambda"']
\end{tikzcd}$$

Then, considering that $\text{untag}_s \circ \text{tag}_s$ is the identity on $LT\mathcal{S}$, and that $\text{tag}_s \circ \text{untag}_s$ is the restriction idempotent of untag_s and therefore below the identity by definition, the pair $(\text{tag}_s, \text{untag}_s)$ is an embedding-projection pair[2].

At this point, it is important to note that this construction provides a highly intuitive representation of terms, which relies very naturally on pretty much all of the features of a bimonoidal join inverse category. Following this observation, one can argue that the categorical structure of bimonoidal join inverse categories is not just sufficient for modelling Rfun, but also necessary for enabling a standard term construction.

[2] tag is a restriction monic (with retraction untag, splitting the restriction idempotent of untag), and in join restriction categories, restriction monics correspond 1-to-1 to embeddings of the induced CPO-enrichment.

Finally, it feels natural to conclude this section by the following observation: if one sees $\mathcal{S}$ as a special case of polynomial functors [25, §3.2, Ex. 3.5], then the tagging operation corresponds to the construction for trees with a specific root.

3.3 Symbols, values and assertions

With a construction of the object of symbols $\mathcal{S}$ and $\mathcal{S}$-valued terms $T\mathcal{S}$, we are finally ready to consider suitable representations of values in full. Recall that symbols are morphisms $1 \xrightarrow{s} \mathcal{S}$ with assertions $\mathcal{S} \xrightarrow{\text{asrt}_s} 1$ as partial inverses.

Given a symbol s, it is clear that it may be lifted to a term consisting of just s with no children. This lifting is given by the canonical morphism $\mathcal{S} \xrightarrow{\text{lift}} T\mathcal{S}$ defined by the composition

$$\begin{tikzcd}
\mathcal{S} \arrow[rr, "\text{lift}"] \arrow[d, "\rho^\dagger"'] & & T\mathcal{S} \\
\mathcal{S} \otimes 1 \arrow[r, "\mathcal{S} \otimes \amalg_1"'] & \mathcal{S} \otimes (1 \oplus (T\mathcal{S} \otimes LT\mathcal{S})) \arrow[r, "\mathcal{S} \otimes \text{fold}_L"'] & \mathcal{S} \otimes LT\mathcal{S} \arrow[u, "\text{fold}_T"']
\end{tikzcd}$$

This gives a denotation of the simplest of terms – the ones consisting of just a symbol s with no children – as compositions $[\![s]\!]_q^\sigma : 1 \xrightarrow{s} \mathcal{S} \xrightarrow{\text{lift}} T\mathcal{S}$ with corresponding assertions $T\mathcal{S} \xrightarrow{\text{lift}^\dagger} \mathcal{S} \xrightarrow{\text{asrt}_s} 1$. More subtly, however, it also gives a representation of arbitrary terms, given by describing their construction. Note first that there is a canonical morphism $1 \xrightarrow{\text{nil}} LT\mathcal{S}$ given by $1 \xrightarrow{\amalg_1} 1 \oplus (T\mathcal{S} \otimes LT\mathcal{S}) \xrightarrow{\text{fold}_L} LT\mathcal{S}$. With this, we can construct the map $\text{cons}_c(s_1, \ldots, s_{n+1}) : 1 \to T\mathcal{S}$ (where $s_i : 1 \to T\mathcal{S}$ for every i) in the following commuting diagram, where the map iter_n is defined by induction: $\text{iter}_0 = \text{id}_{1 \otimes LT\mathcal{S}}$ and $\text{iter}_{n+1} = (T\mathcal{S}^{\otimes n} \otimes \text{fold}_L) \circ (T\mathcal{S}^{\otimes n} \otimes \amalg_2) \circ \text{iter}_n$ for every $n \in \mathbb{N}$.

$$\begin{tikzcd}
1 \arrow[r, "\text{cons}_c(s_1, \ldots, s_n)"] \arrow[d, "\wr\!\|"'] & T\mathcal{S} \\
1^{\otimes n+1} \otimes 1 \arrow[d, "s_1 \otimes \cdots \otimes s_{n+1} \otimes \text{nil}"'] & LT\mathcal{S} \arrow[u, "\text{tag}_c"'] \\
T\mathcal{S}^{\otimes n+1} \otimes LT\mathcal{S} \arrow[d, "T\mathcal{S}^{\otimes n} \otimes \amalg_2"'] & 1 \oplus (T\mathcal{S} \otimes LT\mathcal{S}) \arrow[u, "\text{fold}_L"'] \\
T\mathcal{S}^{\otimes n} \otimes (1 \oplus (T\mathcal{S} \otimes LT\mathcal{S})) \arrow[r, "T\mathcal{S}^{\otimes n} \otimes \text{fold}_L"'] & T\mathcal{S}^{\otimes n} \otimes LT\mathcal{S} \arrow[u, "\text{iter}_n"']
\end{tikzcd}$$

In short: $[\![c(l_1, \ldots, l_n)]\!]_q^\sigma = \text{cons}_c([\![l_1]\!]_q^{\sigma_1}, \ldots, [\![l_n]\!]_q^{\sigma_n}) : 1 \to T\mathcal{S}$ where $\sigma = \uplus_{i=1}^n \sigma_i$. Alternatively, cons_c can be seen as a map $T\mathcal{S}^{\otimes n} \to T\mathcal{S}$ which takes n left-expressions of Rfun and gives back a left-expression labelled by c.

4 Duplication/equality operator and case expressions

In this section, we proceed to unveil the categorical constructions by which one can denote the duplication/equality operator and the case expressions of the language Rfun. For that purpose, in Section 4.1, we establish in inverse category theory a categorical property, called decidable equality, under which we guarantee the existence of the well-behaved denotations of duplication/equality operations, conceptualized as maps $\text{dupeq}_X : X \oplus (X \otimes X) \to X \oplus (X \otimes X)$

for the objects X. Then, we proceed to demonstrate in Section 4.2 that it is sufficient to assume that the object 1 has decidable equality to obtain decidable equality for all objects relevant to the denotational semantics. Finally in Section 4.3, we provide a proof that decidable equality implies decidable pattern matching, a categorical property which ensures that one can construct well-behaved denotations of case expressions. The construction of the denotations of left-expressions $\lfloor v \rfloor$ and case expressions is detailed in Appendix B.1.

4.1 Decidability in restriction category theory

In set theory, a set X is decidable (or has decidable equality) if any pair of elements is either equal or different. This notion has the following analogue in restriction category theory.

▸ **Definition 14.** In an inverse category with inverse products and (finite) joins, one says that an object X has *decidable equality* if the restriction idempotent of the inverse of the duplication map $\overline{\Delta_X^\dagger}$ on X is complemented: that is, if there is an orthocomplement restriction idempotent $\overline{\Delta_X^\dagger}^\perp$ such that $\overline{\Delta_X^\dagger} \vee \overline{\Delta_X^\dagger}^\perp = \mathrm{id}_{X\otimes X}$ and $\overline{\Delta_X^\dagger} \circ \overline{\Delta_X^\dagger}^\perp = 0_{X\otimes X}$.

We say that parallel morphisms in an inverse category f, g are *disjoint*, writing $f \perp g$ iff $f \circ \overline{g} = g \circ \overline{f} = 0$ and likewise for their partial inverses. Note that this is equivalent to saying that $\overline{f} \circ \overline{g} = 0$, and strictly stronger than join compatibility as $f \asymp g$ iff $f \circ \overline{g} = g \circ \overline{f}$ and likewise for their partial inverses.

▸ Proposition 15. In an inverse category with inverse products and (finite) joins, every object X with decidable equality admits *duplication-equality* by a natural involutive automorphism $\mathrm{dupeq}_X : X \oplus (X \otimes X) \to X \oplus (X \otimes X)$.

Proof. See Appendix A.2 for the categorical construction of the map dupeq_X. ◂

In addition, say that an object X has *decidable pattern matching* if any value assertion on X (*i.e.*, any morphism of the form $\overline{s^\dagger} : X \to X$ for a total $s : 1 \to X$) has a complement restriction idempotent $\overline{s^\dagger}^\perp : X \to X$ such that $\overline{s^\dagger} \circ \overline{s^\dagger}^\perp = 0_{X,X}$ and $\overline{s^\dagger} \vee \overline{s^\dagger}^\perp = \mathrm{id}_{X,X}$.

And although assuming that the object $T\mathcal{S}$ has decidable equality is a reasonable assumption given that this is the case – even constructively – in the categories **PInj** and $\mathbf{Set}_*$, it is good practice in an axiomatization to assume the least about the categorical models that we intend to use. For that purpose, we assume in this paper that the unit 1 has decidable equality and observe that the zero element 0 has decidable equality by definition. Then, it is easy to see that $X_1 \otimes X_2$ has decidable equality iff X_1 and X_2 have decidable equality.

In fact, it can be proven that objects formed from the operators 0, 1, $\otimes$, $\oplus$ and μ have decidable equality and pattern matching when the unit 1 does. This is a very mild assumption on the unit 1, as we only need to assume that it is *trivial* in the sense that the coduplicator coincides with the unitors.

4.2 Decidable equality

In this section, it is proven that objects formed from the operators 0, 1, $\otimes$, $\oplus$ and μ have decidable equality when 1 has. In more technical terms, we are going to show that if the †-Frobenius semialgebra $(1, \Delta_1^\dagger)$ is complemented, then so does every †-Frobenius semialgebra $(X, \Delta_X^\dagger)$ for which X is formed from the operators 0, 1, $\otimes$, $\oplus$ and μ.

Establishing this result requires the introduction of the following terminology. Recall that an inverse product is a monoidal tensor $-\otimes-$ with a natural transformation $\Delta_X : X \to X\otimes X$ such that $(X, \Delta_X^\dagger)$ forms a special commutative †-Frobenius semialgebra in each object X.

▸ **Definition 16.** One says that the unit 1 of an inverse product is *trivial* if the coduplicator $\Delta_1^\dagger : 1 \otimes 1 \to 1$ coincides with the unitors $\lambda : 1 \otimes 1 \to 1$ and $\rho : 1 \otimes 1 \to 1$.

▸ **Definition 17.** In an inverse (restriction) category which has (at least finite) joins, one says that a morphism $f : A \to B$ is *domain complemented* – or simply *complemented* – if there exists a restriction idempotent $\overline{f}^{\perp} : A \to A$ satisfying $\overline{f} \circ \overline{f}^{\perp} = 0_{A,A}$ (the empty join) and $\overline{f} \vee \overline{f}^{\perp} = 1_{A,A}$.

Then, in an inverse category which has (at least finite) joins and inverse products, one says that the †-Frobenius (semi)algebra $(X, \Delta_X^\dagger)$ is complemented if the multiplication $\Delta_X^\dagger : X \otimes X \to X$ is.

▸ Proposition 18. In an inverse category with inverse products with a trivial unit, the triple $(1, \Delta_1^\dagger)$ is a special, cospecial, commutative, and complemented †-Frobenius algebra.

Proof. Follows from the definition of the inverse product (see Appendix A.3). ◂

From there, we call a functor *affine* when it is formed from the operators $0, 1, \otimes$ and $\oplus$. In other words, affine functors are the functors generated by induction on the grammar

$$F, G ::= 0 \mid 1 \mid F \otimes G \mid F \oplus G$$

From there, one can show the following proposition.

▸ Proposition 19. Let $F : \mathbf{C} \to \mathbf{C}$ be an inverse affine functor on a bimonoidal join inverse category $\mathbf{C}$, and let the (special, commutative) †-Frobenius semialgebra $(X, \Delta_X^\dagger)$ be complemented. Then the (special, commutative) †-Frobenius semialgebra $(FX, \Delta_{FX}^\dagger)$ is complemented as well.

Proof. By induction on the structure of F. See Appendix A.4 for a detailed proof. ◂

Now, we can proceed to show that our fixpoint construction (via algebraic compactness) is complemented.

▸ Proposition 20. Let $F : \mathbf{C} \to \mathbf{C}$ be an inverse affine functor on a bimonoidal join inverse category $\mathbf{C}$, and consider its unique fixed point μF. Then the (special, commutative) †-Frobenius semialgebra $(\mu F, \Delta_{\mu F}^\dagger)$ is complemented.

Proof. By the construction of fixpoint by algebraic compactness [2,6] (see Appendix A.5). ◂

Combining Proposition 18, 19 and 20, one can construct an orthocomplement for every †-Frobenius semialgebra $(X, \Delta_X^\dagger)$ such that the object X is formed from the operators 0, 1, $\otimes$, $\oplus$ and μ. And therefore, objects formed from the operators 0, 1, $\otimes$, $\oplus$ and μ have decidable equality when the object 1 has.

4.3 Decidable pattern matching

In this section, it is proven that decidable equality implies decidable pattern matching. In more technical terms, we are going to show that for every object X formed from the operators $0, 1, \otimes, \oplus$ and μ, the pseudounital †-Frobenius semialgebra (X, Δ_X, η_X) is discrete whenever the †-Frobenius semialgebra (X, Δ_X) is complemented.

▸ **Definition 21.** A *pseudounital* †-Frobenius semialgebra (X, μ, ξ) is a †-Frobenius semialgebra (X, μ) equipped with a distinguished map $\xi : 1 \to X$ making the diagrams below commute.

$$\begin{tikzcd}
X \arrow[r, "\overline{\xi^\dagger}"] \arrow[d, "\rho^{-1}"'] & X \\
X \otimes 1 \arrow[r, "X \otimes \xi"'] & X \otimes X \arrow[u, "\mu"']
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "\overline{\xi^\dagger}"] \arrow[d, "\mu^\dagger"'] & X \\
X \otimes X \arrow[r, "\xi^\dagger \otimes X"'] & 1 \otimes X \arrow[u, "\lambda"']
\end{tikzcd}$$

Now, let's consider that (X, μ, ξ) is *discrete* if $\mu^\dagger$ and $\xi^\dagger$ are complemented.

▸ **Lemma 22.** *Any †-Frobenius algebra (X, μ, η) with a total counit η,* i.e. *with $\overline{\eta^\dagger} = \mathrm{id}_{X \otimes X}$, is a pseudounital †-Frobenius semialgebra with pseudounit η, and discrete when μ is complemented.*

From this follows that $(0, \Delta_0^\dagger, 0_{1,0})$ is discrete as a †-Frobenius semialgebra, and that $(1, \Delta_1^\dagger, \mathrm{id}_1)$ is so as well whenever it is a †-Frobenius algebra.

▸ Proposition 23. Let $(X, \Delta_X^\dagger, \xi_X)$ and $(Y, \Delta_Y^\dagger, \xi_Y)$ be discrete †-Frobenius semialgebras. Then so are

(i) the product semialgebra $(X \otimes Y, \Delta_{X\otimes Y}^\dagger, \xi_X \otimes \xi_Y \circ \lambda^{-1})$
(ii) the sum semialgebras $(X \oplus Y, \Delta_{X\oplus Y}^\dagger, \amalg_1 \circ \xi_X)$ and $(X \oplus Y, \Delta_{X\oplus Y}^\dagger, \amalg_2 \circ \xi_Y)$

Proof. Since we have previously shown that the coduplicators are complemented, it suffices only to show that the postulated pseudounits satisfy the pseudounit conditions, and that they are complemented. See Appendix A.6 for a detailed proof. ◂

▸ Proposition 24. Let $F : \mathbf{C} \to \mathbf{C}$ be an inverse affine functor on a bimonoidal join inverse category $\mathbf{C}$, and let each (special, commutative) †-Frobenius semialgebra $(F^n(0), \Delta_{F^n(0)}^\dagger, \xi_{F^n(0)})$ be discrete. Then so is $(\mu F, \Delta_{\mu F}^\dagger, \iota_n \circ \xi_{F^n(0)})$.

Proof. We have $\overline{(\iota_n \circ \xi_{F^n(0)})^\dagger} = \overline{\xi_{F^n(0)}^\dagger \circ \iota_n^\dagger} = \iota_n^\dagger \circ \overline{\xi_{F^n(0)}^\dagger} \circ \iota_n$ and therefore pseudounitality follows by commutativity of pseudounit diagram below, noting that the counit diagram is analogous.

$$\begin{tikzcd}
\mu F \arrow[r, "\iota_n^\dagger"] \arrow[d, "\rho^{-1}"'] & F^n(0) \arrow[r, "\overline{\xi_{F^n(0)}^\dagger}"] \arrow[d, "\rho^{-1}"] & F^n(0) \arrow[r, "\iota_n"] & \mu F \\
\mu F \otimes 1 \arrow[r, "\iota_n^\dagger \otimes 1"'] & F^n(0) \otimes 1 \arrow[r, "F^n(0) \otimes \xi_{F^n(0)}"'] & F^n(0) \otimes F^n(0) \arrow[r, "\iota_n \otimes \iota_n"'] \arrow[u, "\Delta_{F^n(0)}^\dagger"] & \mu F \otimes \mu F \arrow[u, "\Delta_{\mu F}^\dagger"']
\end{tikzcd}$$

◂

As a corollary, any †-Frobenius semialgebra constructed using $0, 1, \otimes, \oplus$ and fixed points is discrete.

5 Categorical models of reversible computing

In conclusion, taking our definitions and categorical constructions into account, we define categorical models of reversible computing as follows.

▸ **Definition 25.** A *categorical model of reversible computing* is a bimonoidal join inverse category with decidable equality (for $T\mathcal{S}$).

In summary, we have introduced join inverse categories and constructed the categorical semantics of the instructions of the language Rfun. With arguably weak categorical assumptions, we have shown the strength of join inverse category theory in the semantical study of reversible programming, leading us to the following adequacy theorem (proven in Appendix B.2).

▸ **Theorem 26.** *In any categorical model of reversible computing and under any programming context, a program execution terminates if and only if its denotation is a non-trivial total map.*

References

1 Samson Abramsky and Dominic Horsman. DEMONIC programming: a computational language for single-particle equilibrium thermodynamics, and its formal semantics. In Chris Heunen, Peter Selinger, and Jamie Vicary, editors, *Proceedings 12th International Workshop on Quantum Physics and Logic*, pages 1–16, 2015.

2 Jiří Adámek. Recursive data types in algebraically ω-complete categories. *Information and Computation*, 118:181–190, 1995.

3 Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 249–258, 2005.

4 Holger Bock Axelsen and Robert Glück. What do reversible programs compute? In Martin Hofmann, editor, *FoSSaCS 2011*, volume 6604 of *LNCS*, pages 42–56. Springer, 2011.

5 Holger Bock Axelsen and Robin Kaarsgaard. Join inverse categories as models of reversible recursion. In *Foundations of Software Science and Computation Structures 2016*, volume 9634 of *LNCS*, pages 73–90. Springer, 2016.

6 Michael Barr. Algebraically compact functors. *Journal of Pure and Applied Algebra*, 82(3):211–231, 1992.

7 Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

8 Antoine Bérut, Artak Arakelyan, Artyom Petrosyan, Sergio Ciliberto, Raoul Dillenschneider, and Eric Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483(7388):187–189, 2012.

9 J. R. B. Cockett and Stephen Lack. Restriction categories I: Categories of partial maps. *Theoretical Computer Science*, 270(1–2):223–259, 2002.

10 J. Robin B. Cockett and Stephen Lack. Restriction categories II: Partial map classification. *Theoretical Computer Science*, 294(1):61–102, 2003.

11 Robin Cockett and Stephen Lack. Restriction categories III: Colimits, partial limits and extensivity. *Mathematical Structures in Computer Science*, 17(04):775–817, 2007.

12 Ioana Cristescu, Jean Krivine, and Daniele Varacca. A compositional semantics for the reversible p-calculus. In *Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium on*, pages 388–397. IEEE, 2013.

13 Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3-4):219–253, 1982.

14 Brett Gordon Giles. *An investigation of some theoretical aspects of reversible computing.* PhD thesis, University of Calgary, 2014.

15 Xiuzhan Guo. *Products, Joins, Meets, and Ranges in Restriction Categories.* PhD thesis, University of Calgary, 2012.

16 Peter Mark Hines. *The Algebra of Self-Similarity and its Applications*. PhD thesis, University of Wales, Bangor, 1998.

17 Roshan P. James and Amr Sabry. Theseus: A high level language for reversible computing, 2014. Reversible Computing 2014.

18 Robin Kaarsgaard, Holger Bock Axelsen, and Robert Glück. Join inverse categories and reversible recursion. *Journal of Logical and Algebraic Methods in Programming*, 87:33–50, 2017.

19 J. Kastl. Inverse categories. In Hans-Jürgen Hoehnke, editor, *Algebraische Modelle, Kategorien und Gruppoide*, pages 51–60. Akademie Verlag, Berlin, 1979.

20 Martin Kutrib and Andreas Malcher. Reversible pushdown automata. In A.-H. Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA 2010*, volume 6031 of *LNCS*, pages 368–379. Springer-Verlag, 2010.

21 Martin Kutrib and Matthias Wendlandt. Reversible limited automata. In J. Durand-Lose and B. Nagy, editors, *MCU 2015*, volume 9288 of *LNCS*, pages 113–128. Springer, 2015.

22 Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM journal of research and development*, 5(3):183–191, 1961.

23 Mark V Lawson. *Inverse Semigroups: The Theory of Partial Symmetries*. World Scientific, 1998.

24 Octavio Malherbe, Philip Scott, and Peter Selinger. Presheaf models of quantum computation: an outline. In *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*, pages 178–194. Springer, 2013.

25 Ieke Moerdijk and Erik Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1):189–218, 2000.

26 Gordon E Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff. *IEEE Solid-State Circuits Newsletter*, 3(20):33–35, 2006.

27 Mathys Rennela and Sam Staton. Complete positivity and natural representation of quantum computations. In *MFPS XXXI*, volume 319, pages 369–385. Electronic Notes in Theoretical Computer Science, 2015.

28 Mathys Rennela, Sam Staton, and Robert Furber. Infinite-dimensionality in quantum foundations: W*-algebras as presheaves over matrix algebras. In *QPL'16*, volume 236 of *Electronic Proceedings in Theoretical Computer Science*, pages 161–173. Open Publishing Association, 2017.

29 Markus Schordan, David Jefferson, Peter Barnes, Tomas Oppelstrup, and Daniel Quinlan. Reverse code generation for parallel discrete event simulation. In Jean Krivine and Jean-Bernard Stefani, editors, *RC 2015*, volume 9138 of *LNCS*, pages 95–110. Springer, 2015.

30 Ulrik Schultz, Mirko Bordignon, and Kasper Stoy. Robust and reversible execution of self-reconfiguration sequences. *Robotica*, 29(01):35–57, 2011.

31 Ulrik Pagh Schultz, Johan Sund Laursen, Lars-Peter Ellekilde, and Holger Bock Axelsen. Towards a domain-specific language for reversible assembly sequences. In *Reversible Computation*, pages 111–126. Springer, 2015.

32 Tommaso Toffoli. Reversible Computing. In *Proceedings of the Colloquium on Automata, Languages and Programming*, pages 632–644. Springer Verlag, 1980.

33 Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Towards a reversible functional language. In Alexis De Vos and Robert Wille, editors, *RC 2011*, volume 7165 of *LNCS*, pages 14–29. Springer, 2012.

34 Tetsuo Yokoyama and Robert Glück. A reversible programming language and its invertible self-interpreter. In *PEPM '07, Proceedings*, pages 144–153. ACM, 2007.

A Omitted proofs

A.1 Proof of Theorem 13

In the following, recall that $\mathrm{Inv}(\mathbf{C})$ for a restriction category $\mathbf{C}$ is the *cofree inverse category*, *i.e.*, the largest inverse subcategory of $\mathbf{C}$; it is given on objects by objects of $\mathbf{C}$, and on morphisms by all partial isomorphisms (*i.e.*, morphisms f for which there is a map f^* such that $f \circ f^* = \overline{f^*}$ and $f^* \circ f = \overline{f}$) of $\mathbf{C}$. Similarly, $\mathrm{Total}(\mathbf{C})$ is the subcategory of total morphisms of $\mathbf{C}$, given on objects by objects of $\mathbf{C}$, and on morphisms by total morphisms (*i.e.*, morphisms satisfying $\overline{f} = \mathrm{id}$) of $\mathbf{C}$. For a restriction category $\mathbf{C}$, $\mathrm{Split}(\mathbf{C})$ denotes the idempotent splitting ("Karoubi envelope") of $\mathbf{C}$ with respect to all *restriction idempotents*, *i.e.*, morphisms e such that $\overline{e} = e$.

Let $\mathbf{C}$ be an inverse category, and let $\mathcal{N}$ denote the least stable system of monics containing both $\widehat{\mathcal{M}_{\text{gap}}}$ (as in [15]) and all duplicator monics $\langle \mathrm{id}, \mathrm{id} \rangle : X \to X \times X$ in $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$, and $\breve{\mathbf{C}}$ the resulting category of partial maps $\mathrm{Par}(\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}, \mathcal{N})$.

The fact that the category $\mathbf{C}$ embeds faithfully in the category $\mathrm{Inv}(\breve{\mathbf{C}})$, which is join inverse and algebraically ω-compact for join restriction functors, was previously shown in [15] and [5] respectively.

As for (iii), we note that $\breve{\mathbf{C}}$ is classified (see, *e.g.*, [10]), and since the category $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$ has products respectively coproducts (as it is a topos), it follows by [15] respectively [10] that the category $\breve{\mathbf{C}}$ has all finite restriction products respectively restriction coproducts, and that restriction products are *discrete* since all diagonal maps are in $\mathcal{N}$ (see [14]).

Further, restriction products and coproducts distribute, and the restriction zero annihilates restriction products, as they are constructed precisely by the products and coproducts of the category $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$.

Since joins in the category $\breve{\mathbf{C}}$ are constructed as stable colimits of certain pullbacks (specifically the *$\mathcal{M}$-amalgamable diagrams*) in the category $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$, to show that they are join preserving, it suffices to show that products and coproducts in the category $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$ preserve colimits of pullbacks (as all colimits are stable in a topos). That the restriction product functors $- \times X$ (and symmetrically) in the category $\breve{\mathbf{C}}$ preserve joins thus follows by the fact that they are products in the category $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$ (specifically left adjoints), so they preserve colimits (by Freyd's adjoint functor theorem), and commute with arbitrary limits, specifically pullbacks. Similarly, that the same is the case for restriction coproduct functors $- + X$ (and symmetrically) in the category $\breve{\mathbf{C}}$ follows likewise by the fact that they are coproducts in the category $\mathbf{Set}^{\mathrm{Total}(\mathrm{Split}(\mathbf{C}))^{\mathrm{op}}}$, and therefore are stable under pullback (*i.e.*, commute with pullbacks) and commute with arbitrary colimits.

Finally, since the category $\mathrm{Inv}(\mathbf{C})$ is a join inverse category whenever the category $\mathbf{C}$ is a join restriction category (see, *e.g.*, [15]), it remains only to show that when the category $\mathbf{C}$ is a distributive restriction category with $A \times 0 \cong 0$ for all A, then the category $\mathrm{Inv}(\mathbf{C})$ is a bimonoidal inverse category. Since the category $\breve{\mathbf{C}}$ has discrete restriction products, it follows by [14] that these are inverse products in the category $\mathrm{Inv}(\breve{\mathbf{C}})$. Further, restriction coproducts in the category $\breve{\mathbf{C}}$ define a disjointness tensor in the category $\mathrm{Inv}(\breve{\mathbf{C}})$ as any restriction coproduct has the restriction zero as unit (when it exists), the bifunctor $- + -$ be monoidal (when it has a unit) and restriction preserving, and jointly epic coproduct injections (which are further partial isomorphisms, so preserved in the category $\mathrm{Inv}(\breve{\mathbf{C}})$). Since distributivity and annihilation are isomorphisms, these are preserved in the category $\mathrm{Inv}(\breve{\mathbf{C}})$.

A.2 Proof of Proposition 15

We construct $\mathrm{dupeq}_X = d_{1,X} \vee d_{2,X} \vee d_{3,X}$ where the maps $d_{i,X}$ are defined as follows:

$$\begin{tikzcd}
X \oplus (X \otimes X) \arrow[r, "d_{1,X}"] \arrow[d, "\amalg_2^\dagger"'] & X \oplus (X \otimes X) \\
X \otimes X \arrow[r, "\Delta^\dagger"'] & X \arrow[u, "\amalg_1"']
\end{tikzcd}
\quad
\begin{tikzcd}
X \oplus (X \otimes X) \arrow[r, "d_{2,X}"] \arrow[d, "\amalg_2^\dagger"'] & X \oplus (X \otimes X) \\
X \otimes X \arrow[r, "\overline{\Delta^\dagger}^{\perp}"'] & X \otimes X \arrow[u, "\amalg_2"']
\end{tikzcd}
\quad
\begin{tikzcd}
X \oplus (X \otimes X) \arrow[r, "d_{3,X}"] \arrow[d, "\amalg_1^\dagger"'] & X \oplus (X \otimes X) \\
X \arrow[r, "\Delta"'] & X \otimes X \arrow[u, "\amalg_2"']
\end{tikzcd}$$

Clearly, one can observe that $d_1 \perp d_3$ and $d_2 \perp d_3$ by $\amalg_1^\dagger \perp \amalg_2^\dagger$, and $d_1 \perp d_2$ by $\Delta_X^\dagger = \Delta_X^\dagger \circ \overline{\Delta_X^\dagger}$ and $\overline{\Delta_X^\dagger} \perp \overline{\Delta_X^\dagger}^{\perp}$. In the other direction, we notice that $d_{1,X}^\dagger = d_{3,X}$, $d_{2,X}^\dagger = d_{2,X}$, $d_{3,X}^\dagger = d_{1,X}$. Thus they are pairwise disjoint by the exact same arguments as above, so the set $\{d_{1,X}, d_{2,X}, d_{3,X}\}$ is a compatible set, and in turn their join $\mathrm{dupeq}_X = d_{1,X} \vee d_{2,X} \vee d_{3,X}$ exists.

Notice also that this makes dupeq_X and involution, as

$$\begin{aligned}
\mathrm{dupeq}_X^\dagger &= (d_{1,X} \vee d_{2,X} \vee d_{3,X})^\dagger = d_{1,X}^\dagger \vee d_{2,X}^\dagger \vee d_{3,X}^\dagger \\
&= d_{3,X} \vee d_{2,X} \vee d_{1,X} = d_{1,X} \vee d_{2,X} \vee d_{3,X} = \mathrm{dupeq}_X
\end{aligned}$$

To see that it is an automorphism, it suffices to show that $\overline{\mathrm{dupeq}_X} = \mathrm{id}_{X \oplus (X \otimes X)}$.

Computing the restriction idempotents of the individual components (using the fact that $\overline{f \circ g} = \overline{\overline{f} \circ g}$),

$$\begin{aligned}
\overline{d_{1,X}} &= \overline{\overline{\overline{\amalg_1} \circ \Delta_X^\dagger} \circ \amalg_2^\dagger} = \overline{\overline{\mathrm{id}_X \circ \Delta_X^\dagger} \circ \amalg_2^\dagger} = \overline{\overline{\Delta_X^\dagger} \circ \amalg_2^\dagger} \\
\overline{d_{2,X}} &= \overline{\overline{\overline{\amalg_2} \circ \overline{\Delta_X^\dagger}^{\perp}} \circ \amalg_2^\dagger} = \overline{\overline{\mathrm{id}_{X \otimes X} \circ \overline{\Delta_X^\dagger}^{\perp}} \circ \amalg_2^\dagger} = \overline{\overline{\Delta_X^\dagger}^{\perp} \circ \amalg_2^\dagger} \\
\overline{d_{3,X}} &= \overline{\overline{\overline{\amalg_2} \circ \Delta_X} \circ \amalg_1^\dagger} = \overline{\overline{\mathrm{id}_{X \otimes X} \circ \Delta_X} \circ \amalg_1^\dagger} = \overline{\overline{\Delta_X} \circ \amalg_1^\dagger} = \overline{\mathrm{id}_X \circ \amalg_1^\dagger} = \overline{\amalg_1^\dagger}.
\end{aligned}$$

But then

$$\begin{aligned}
\overline{d_{1,X} \vee d_{2,X} \vee d_{3,X}} &= \overline{d_{1,X}} \vee \overline{d_{2,X}} \vee \overline{d_{3,X}} = \overline{\overline{\Delta_X^\dagger} \circ \amalg_2^\dagger} \vee \overline{\overline{\Delta_X^\dagger}^{\perp} \circ \amalg_2^\dagger} \vee \overline{\amalg_1^\dagger} \\
&= \overline{\overline{\Delta_X^\dagger} \circ \amalg_2^\dagger \vee \overline{\Delta_X^\dagger}^{\perp} \circ \amalg_2^\dagger} \vee \overline{\amalg_1^\dagger} \\
&= \overline{(\overline{\Delta_X^\dagger} \vee \overline{\Delta_X^\dagger}^{\perp}) \circ \amalg_2^\dagger} \vee \overline{\amalg_1^\dagger} \\
&= \overline{\mathrm{id}_{X \otimes X} \circ \amalg_2^\dagger} \vee \overline{\amalg_1^\dagger} \\
&= \overline{\amalg_2^\dagger} \vee \overline{\amalg_1^\dagger} = \overline{\amalg_2^\dagger \vee \amalg_1^\dagger} = \mathrm{id}_{X \oplus (X \otimes X)}
\end{aligned}$$

so dupeq_X is total, which was what we wanted.

A.3 Proof of Proposition 18

By the inverse product, we have that the triple $(1, \Delta_1, \Delta_1^\dagger)$ is a (special, commutative) †-Frobenius semialgebra, so suffices to show that id_1 serves as unit and counit, and that the resulting †-Frobenius algebra is cospecial and complemented. Commutativity of the (co)unitor diagram for the (co)monoid

$$\begin{tikzcd}
1 \otimes 1 \arrow[r, "\mathrm{id} \otimes \mathrm{id}"] \arrow[dr, "\rho"'] & 1 \otimes 1 \arrow[d, "\Delta_1^\dagger"'] & 1 \otimes 1 \arrow[l, "\mathrm{id} \otimes \mathrm{id}"'] \arrow[dl, "\lambda"] \\
 & 1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \otimes 1 & 1 \otimes 1 \arrow[l, "\mathrm{id} \otimes \mathrm{id}"'] \arrow[r, "\mathrm{id} \otimes \mathrm{id}"] & 1 \otimes 1 \\
 & 1 \arrow[ul, "\rho^\dagger"] \arrow[u, "\Delta_1"] \arrow[ur, "\lambda^\dagger"'] &
\end{tikzcd}$$

follows precisely by the assumption that 1 is trivial, and the unit and counit diagrams for †-Frobenius algebras reduces to showing that $\mathrm{id} = \Delta_1 \circ \lambda = \rho \circ \Delta_1^\dagger$, which follows by triviality again. Cospecialness follows by $\Delta_1^\dagger = \rho$ so $\overline{\Delta_1^\dagger} = \overline{\rho} = \mathrm{id}$, and hence by Δ_1 total with a total inverse, it is an isomorphism. Finally, since $\overline{\Delta_1^\dagger} = \mathrm{id}$, the only choice for $\overline{\Delta_1^\dagger}^\perp$ is the zero morphism $0_{1\otimes 1, 1\otimes 1}$ which satisfies the conditions.

A.4 Proof of Proposition 19

In the following, we reason by induction on the structure of F.

- Case $F(X) = 0$. By annihilation, we observe that $0 \otimes 1 \cong 1 \otimes 0 \cong 0$ and therefore $(0, \Delta_0, \Delta_0^\dagger)$ is a (special, cospecial, commutative, complemented) †-Frobenius algebra with unit $0_{1,0}$ by argument analogous to that of Prop. 18.
- Case $F(X) = 1$. By Prop. 18.
- Case $F(X) = F_1(X) \otimes F_2(X)$. We write $X_i \overset{\mathrm{def}}{=} F_i(X)$. By induction, $(X_1, \Delta_{X_1}, \Delta_{X_1}^\dagger)$ and $(X_2, \Delta_{X_2}, \Delta_{X_2}^\dagger)$ are complemented (special, commutative) †-Frobenius semialgebras. By commutativity of the diagram (see [14])

$$\begin{tikzcd}
X_1 \otimes X_2 \arrow[rr, "\Delta_{X_1} \otimes \Delta_{X_2}"] \arrow[dr, "\Delta_{X_1 \otimes X_2}"'] & & X_1 \otimes X_1 \otimes X_2 \otimes X_2 \arrow[dl, "X_1 \otimes \gamma_{X_1, X_2} \otimes X_2"] \\
 & X_1 \otimes X_2 \otimes X_1 \otimes X_2 &
\end{tikzcd}$$

  we obtain that

$$\overline{\Delta^\dagger_{X_1 \otimes X_2}}^\perp = \overline{\overline{\Delta^\dagger_{X_1}}^\perp \otimes \overline{\Delta^\dagger_{X_2}}^\perp \circ X_1 \otimes \gamma \otimes X_2}$$

  which can be seen (after some computation) to satisfy the conditions for a complement.
- Case $F(X) = F_1(X) \oplus F_2(X)$. We write $X_i \overset{\mathrm{def}}{=} F_i(X)$. By induction, $(X_1, \Delta_{X_1}, \Delta_{X_1}^\dagger)$ and $(X_2, \Delta_{X_2}, \Delta_{X_2}^\dagger)$ are complemented (special, commutative) †-Frobenius semialgebras. Then $\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger \leqslant \Delta_{X_1 \oplus X_2}$ since

$$\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger = \amalg_1 \otimes \amalg_1 \circ \amalg_1^\dagger \otimes \amalg_1^\dagger \circ \Delta_{X_1 \oplus X_2} = \overline{\amalg_1^\dagger \otimes \amalg_1^\dagger} \circ \Delta_{X_1 \oplus X_2}$$

$$\text{and} \qquad \Delta_{X_1 \oplus X_2} \circ \overline{\overline{\amalg_1^\dagger \otimes \amalg_1^\dagger} \circ \Delta_{X_1 \oplus X_2}} = \Delta_{X_1 \oplus X_2} \circ \overline{\amalg_1^\dagger \otimes \amalg_1^\dagger \circ \Delta_{X_1 \oplus X_2}} = \overline{\amalg_1^\dagger \otimes \amalg_1^\dagger} \circ \Delta_{X_1 \oplus X_2}$$

  By an entirely analogous argument, $\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger \leqslant \Delta_{X_1 \oplus X_2}$ but then

$$(\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger) \vee (\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger) \leqslant \Delta_{X_1 \oplus X_2}$$

  and since

$$\begin{aligned}
&\overline{(\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger) \vee (\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger)} = \overline{(\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger)} \vee \overline{(\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger)} \\
&= \overline{\overline{\amalg_1 \otimes \amalg_1} \circ \Delta_{X_1} \circ \amalg_1^\dagger} \vee \overline{\overline{\amalg_2 \otimes \amalg_2} \circ \Delta_{X_2} \circ \amalg_2^\dagger} = \overline{\Delta_{X_1} \circ \amalg_1^\dagger} \vee \overline{\Delta_{X_2} \circ \amalg_2^\dagger} \\
&= \overline{\overline{\Delta_{X_1}} \circ \amalg_1^\dagger} \vee \overline{\overline{\Delta_{X_2}} \circ \amalg_2^\dagger} = \overline{\amalg_1^\dagger} \vee \overline{\amalg_2^\dagger} = \mathrm{id}
\end{aligned}$$

by totality of $\amalg_i$ and Δ_{X_1} and Δ_{X_2}, it follows that

$$(\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger) \vee (\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger) = \Delta_{X_1 \oplus X_2}$$

and so

$$\begin{aligned}
&(\amalg_1 \circ \Delta_{X_1}^\dagger \circ \amalg_1^\dagger \otimes \amalg_1^\dagger) \vee (\amalg_2 \circ \Delta_{X_2}^\dagger \circ \amalg_2^\dagger \otimes \amalg_2^\dagger) \\
&\quad = (\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger)^\dagger \vee (\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger)^\dagger \\
&\quad = ((\amalg_1 \otimes \amalg_1 \circ \Delta_{X_1} \circ \amalg_1^\dagger) \vee (\amalg_2 \otimes \amalg_2 \circ \Delta_{X_2} \circ \amalg_2^\dagger))^\dagger \\
&\quad = \Delta_{X_1 \oplus X_2}^\dagger
\end{aligned}$$

giving us

$$\begin{aligned}
\overline{\Delta_{X_1 \oplus X_2}^\dagger} &= \overline{(\amalg_1 \circ \Delta_{X_1}^\dagger \circ \amalg_1^\dagger \otimes \amalg_1^\dagger) \vee (\amalg_2 \circ \Delta_{X_2}^\dagger \circ \amalg_2^\dagger \otimes \amalg_2^\dagger)} \\
&= \overline{\amalg_1 \circ \Delta_{X_1}^\dagger \circ \amalg_1^\dagger \otimes \amalg_1^\dagger} \vee \overline{\amalg_2 \circ \Delta_{X_2}^\dagger \circ \amalg_2^\dagger \otimes \amalg_2^\dagger} \\
&= \overline{\overline{\amalg_1} \circ \Delta_{X_1}^\dagger \circ \amalg_1^\dagger \otimes \amalg_1^\dagger} \vee \overline{\overline{\amalg_2} \circ \Delta_{X_2}^\dagger \circ \amalg_2^\dagger \otimes \amalg_2^\dagger} \\
&= \overline{\Delta_{X_1}^\dagger \circ \amalg_1^\dagger \otimes \amalg_1^\dagger} \vee \overline{\Delta_{X_2}^\dagger \circ \amalg_2^\dagger \otimes \amalg_2^\dagger} = \overline{\overline{\Delta_{X_1}^\dagger} \circ \amalg_1^\dagger \otimes \amalg_1^\dagger} \vee \overline{\overline{\Delta_{X_2}^\dagger} \circ \amalg_2^\dagger \otimes \amalg_2^\dagger}.
\end{aligned}$$

As such, choosing

$$\overline{\Delta_{X \oplus Y}^\dagger}^\perp \overset{\text{def}}{=} \overline{\overline{\Delta_{X_1}^\dagger}^\perp \circ \amalg_1^\dagger \otimes \amalg_1^\dagger} \vee \overline{\overline{\Delta_{X_2}^\dagger}^\perp \circ \amalg_2^\dagger \otimes \amalg_2^\dagger} \vee \overline{\amalg_1^\dagger \otimes \amalg_2^\dagger} \vee \overline{\amalg_2^\dagger \otimes \amalg_1^\dagger}$$

one realizes after lengthy but straightforward computations that this definition is, indeed, the complement of $\overline{\Delta_{X \oplus Y}^\dagger}$.

A.5 Proof of Proposition 20

By [2] (see also [6]), we obtain from algebraic compactness the commuting diagram

$$\begin{tikzcd}
0 \arrow[r, "!"] \arrow[ddr, "\iota_0"'] & F(0) \arrow[r, "F(!)"] \arrow[dd, "\iota_1"] & F^2(0) \arrow[r, "F^2(!)"] \arrow[ddl, "\iota_2"] & \cdots \\
& & & \\
& \mu F \arrow[ddl] \arrow[dd, "\iota_1^\dagger"] \arrow[ddr, "\iota_2^\dagger"] \arrow[ddl, phantom, "\iota_0^\dagger"] & & \\
& & & \\
0 & F(0) \arrow[l, "!^\dagger"] & F^2(0) \arrow[l, "F(!^\dagger)"] & \cdots \arrow[l, "F^2(!^\dagger)"]
\end{tikzcd}$$

where all ι_n are embeddings (so restriction monics) with corresponding projections $\iota_n^\dagger$, further satisfying that $\{\iota_n \circ \iota_n^\dagger\}_{n \in \mathbb{N}} = \{\overline{\iota_n^\dagger}\}_{n \in \mathbb{N}}$ is an ω-chain with supremum $\mathrm{id}_{\mu F}$.

Analogously to the inverse sum case, one can show that $\iota_n \otimes \iota_n \circ \Delta_{F^n(0)} \circ \iota_n^\dagger \leqslant \Delta_{\mu F}$ since

$$\begin{aligned}
\Delta_{\mu F} \circ \overline{\iota_n \otimes \iota_n \circ \Delta_{F^n(0)} \circ \iota_n^\dagger} &= \Delta_{\mu F} \circ \overline{\iota_n \otimes \iota_n \circ \iota_n^\dagger \otimes \iota_n^\dagger \circ \Delta_{\mu F}} \\
&= \Delta_{\mu F} \circ \overline{\overline{\iota_n^\dagger \otimes \iota_n^\dagger} \circ \Delta_{\mu F}} = \Delta_{\mu F} \circ \overline{\iota_n^\dagger \otimes \iota_n^\dagger \circ \Delta_{\mu F}} \\
&= \overline{\iota_n^\dagger \otimes \iota_n^\dagger} \circ \Delta_{\mu F}
\end{aligned}$$

and $\iota_n \otimes \iota_n \circ \Delta_{F^n(0)} \circ \iota_n^\dagger = \iota_n \otimes \iota_n \circ \iota_n^\dagger \otimes \iota_n^\dagger \circ \Delta_{\mu F} = \overline{\iota_n^\dagger \otimes \iota_n^\dagger} \circ \Delta_{\mu F}$.

Since all of these morphisms have a common supremum in the form of $\Delta_{\mu F}$, their join exists and turns out to be precisely $\Delta_{\mu F}$ because

$$\begin{aligned}\bigvee_{n\in\mathbb{N}} \left(\iota_n \otimes \iota_n \circ \Delta_{F^n(0)} \circ \iota_n^\dagger\right) &= \bigvee_{n\in\mathbb{N}} \left(\overline{\iota_n^\dagger \otimes \iota_n^\dagger} \circ \Delta_{\mu F}\right) = \left(\bigvee_{n\in\mathbb{N}} \overline{\iota_n^\dagger \otimes \iota_n^\dagger}\right) \circ \Delta_{\mu F} \\ &= \left(\bigvee_{n\in\mathbb{N}} \overline{\iota_n^\dagger} \otimes \overline{\iota_n^\dagger}\right) \circ \Delta_{\mu F} = \left(\bigvee_{n\in\mathbb{N}} \overline{\iota_n^\dagger}\right) \otimes \left(\bigvee_{n\in\mathbb{N}} \overline{\iota_n^\dagger}\right) \circ \Delta_{\mu F} = \mathrm{id}_{\mu F} \otimes \mathrm{id}_{\mu F} \circ \Delta_{\mu_F} = \Delta_{\mu F}.\end{aligned}$$

As such,

$$\begin{aligned}\overline{\Delta_{\mu F}^\dagger} &= \overline{\bigvee_{n\in\mathbb{N}} \left(\iota_n \otimes \iota_n \circ \Delta_{F^n(0)} \circ \iota_n^\dagger\right)^\dagger} = \overline{\bigvee_{n\in\mathbb{N}} \iota_n \circ \Delta_{F^n(0)}^\dagger \circ \iota_n^\dagger \otimes \iota_n^\dagger} = \bigvee_{n\in\mathbb{N}} \overline{\iota_n \circ \Delta_{F^n(0)}^\dagger \circ \iota_n^\dagger \otimes \iota_n^\dagger} \\ &= \bigvee_{n\in\mathbb{N}} \overline{\overline{\overline{\iota_n} \circ \Delta_{F^n(0)}^\dagger} \circ \iota_n^\dagger \otimes \iota_n^\dagger} = \bigvee_{n\in\mathbb{N}} \overline{\overline{\Delta_{F^n(0)}^\dagger} \circ \iota_n^\dagger \otimes \iota_n^\dagger}\end{aligned}$$

Finally, we observe that any embedding arising from an inverse affine functor is either an isomorphism, a zero map, or a quasi-injection $\amalg_i$, so it follows that the restriction idempotent of any projection $\overline{\iota_n^\dagger}$ has a complement (for zero maps, choose the identity; for isomorphisms – which are total – choose the zero map; for $\overline{\amalg_1^\dagger}$ choose $\overline{\amalg_2^\dagger}$ and vice versa).

As such, choosing

$$\overline{\Delta_{F^n(0)}^\dagger}^{\perp} \overset{\mathrm{def}}{=} \bigvee_{n\in\mathbb{N}} \overline{\overline{\Delta_{F^n(0)}^\dagger}^{\perp} \circ \iota_n^\dagger \otimes \iota_n^\dagger} \vee \bigvee_{n\in\mathbb{N}} \overline{\iota_n^\dagger} \otimes \overline{\iota_n^\dagger}^{\perp} \vee \bigvee_{n\in\mathbb{N}} \overline{\iota_n^\dagger}^{\perp} \otimes \overline{\iota_n^\dagger}$$

it follows by lengthy but straightforward computation that this satisfies the requirements for a complement.

A.6 Proof of Proposition 23

Since we have previously shown that the coduplicators are complemented, it suffices only to show that the postulated pseudounits satisfy the pseudounit conditions, and that they are complemented.

For (i), we observe that

$$\overline{(\xi_X \otimes \xi_Y \circ \lambda^{-1})^\dagger} = \overline{\lambda \circ \xi_X^\dagger \otimes \xi_Y^\dagger} = \overline{\overline{\lambda} \circ \xi_X^\dagger \otimes \xi_Y^\dagger} = \overline{\xi_X^\dagger \otimes \xi_Y^\dagger} = \overline{\xi_X^\dagger} \otimes \overline{\xi_Y^\dagger}.$$

and that the unit diagram commutes can be seen from commutativity of the diagram in Fig. 1.

where the inner top trapezoid is what we want to show, and the outermost square commutes precisely by the respective diagrams for the constituent semialgebras.

Commutativity of the counit diagram follows by analogous argument. It follows straightforwardly that $\overline{\xi_X^\dagger}^{\perp} \otimes \overline{\xi_Y^\dagger}^{\perp}$ is the complement of $\overline{\xi_X^\dagger} \otimes \overline{\xi_Y^\dagger}$.

For (ii), we notice that

$$\overline{(\amalg_1 \circ \xi_X)^\dagger} = \overline{\xi_X^\dagger \circ \amalg_1^\dagger} = \amalg_1 \circ \xi_X \circ \xi_X^\dagger \circ \amalg_1^\dagger = \amalg_1 \circ \overline{\xi_X^\dagger} \circ \amalg_1^\dagger$$

and the commutativity of the unit diagram follows by the commutativity of the following diagram

$$\begin{tikzcd}
 & X \otimes 1 \otimes Y \otimes 1 & & X \otimes Y \arrow[ll, "\rho^{-1} \otimes \rho^{-1}"'] \arrow[dl, "\rho^{-1}"] \arrow[ddd, "\overline{\xi_X^\dagger} \otimes \overline{\xi_Y^\dagger}"] \\
 & & X \otimes Y \otimes 1 \arrow[ul, "\rho^{-1} \otimes Y \otimes 1"'] \arrow[d, "X \otimes Y \otimes \lambda^{-1}"'] & \\
X \otimes 1 \otimes Y \otimes 1 \arrow[uur, equal] \arrow[rr, "X \otimes \gamma \otimes 1"] \arrow[ddr, "X \otimes \xi_X \otimes Y \otimes \xi_Y"'] & & X \otimes Y \otimes 1 \otimes 1 \arrow[d, "X \otimes Y \otimes \xi_X \otimes \xi_Y"'] & \\
 & & X \otimes Y \otimes X \otimes Y \arrow[dr, "\Delta_{X \otimes Y}"] & \\
 & X \otimes X \otimes Y \otimes Y \arrow[ur, "X \otimes \gamma \otimes Y"] \arrow[rr, "\Delta_X \otimes \Delta_Y"'] & & X \otimes Y
\end{tikzcd}$$

Figure 1 Commutativity of the pseudounit diagram for inverse products.

$$\begin{tikzcd}
X \oplus Y \arrow[r, "\amalg_1^\dagger"] \arrow[d, "\rho^{-1}"'] & X \arrow[r, "\overline{\xi_X^\dagger}"] \arrow[d, "\rho^{-1}"] & X \arrow[r, "\amalg_1"] & X \oplus Y \\
(X \oplus Y) \otimes 1 \arrow[r, "\amalg_1^\dagger \otimes 1"] \arrow[ddd, "(X \oplus Y) \otimes \xi_X"] & X \otimes 1 \arrow[r, "X \otimes \xi_X"] & X \otimes X \arrow[u, "\Delta_X^\dagger"'] & \\
 & & & (X \oplus Y) \otimes (X \otimes Y) \arrow[uu, "\Delta_{X \oplus Y}^\dagger"'] \\
 & & & \\
(X \oplus Y) \otimes X \arrow[rr, "(X \oplus Y) \otimes \amalg_1"'] & & (X \oplus Y) \otimes (X \otimes Y) \arrow[uu, "\amalg_1^\dagger \otimes \amalg_1^\dagger"] \arrow[uur, "\overline{\amalg_1^\dagger} \otimes \overline{\amalg_1^\dagger}"'] &
\end{tikzcd}$$

where the center top square commutes precisely since it is the unit diagram for $(X, \Delta_X^\dagger, \xi_X)$, and the outer path is what we wanted to show. The unit diagram for $(Y, \Delta_Y^\dagger, \xi_Y)$ is identical (save for subscripts), and the counit diagrams are analogous. Finally, it follows by straightforward computation that $\overline{\overline{\xi_X^\dagger}^{\perp} \circ \amalg_1^\dagger} \vee \overline{\amalg_2^\dagger}$ is the complement of $\overline{\xi_X^\dagger \circ \amalg_1^\dagger}$, and symmetrically that $\overline{\overline{\xi_Y^\dagger}^{\perp} \circ \amalg_2^\dagger} \vee \overline{\amalg_1^\dagger}$ is the complement of $\overline{\xi_Y^\dagger \circ \amalg_2^\dagger}$.

B Rfun: a language for reversible computing

The abstract syntax of the first-order functional language Rfun [33] can be summed up by the following grammars.

$$\begin{aligned}
\text{Left Expressions } l &::= x \mid c(l_1,\dots,l_n)\\
\text{Expressions } e &::= l \mid \textbf{let } l_{out} = f\, l_{in} \textbf{ in } e\\
&\mid \textbf{rlet } l_{in} = f\, l_{out} \textbf{ in } e\\
&\mid \textbf{case}\, l \textbf{ of } \{l_i \to e_i\}_{i=1}^{m}\\
\text{Definitions } d &::= f\, x \triangleq e\\
\text{Programs } q &::= d_1;\dots;d_n
\end{aligned}$$

Contrary to the original presentation of the language, the argument of a definition is here a variable, and not a left expression. We recover the original expressivity of Rfun by introducing some syntactic sugar: definitions $f\, c(l_1,\dots,l_n) \triangleq e$ will stand for the term

$$f\, x \triangleq \textbf{case } x \textbf{ of } c(l_1,\dots,l_n) \to e$$

It is important to recall before going any deeper in the presentation of Rfun that:

- Function and variable identifiers do not belong to the same sort.
- We suppose that programs in the same sequences of definitions have (pairwise) distinct functional identifiers.
- Recursive definitions of functions are not allowed.
- In a left expression, a variable must appear exactly once (see [34]).
- Domains of substitutions are (pairwise) disjoint.

Now a presentation of Rfun's big step operational semantics is given, with expression judgement $\langle q,\sigma\rangle \vdash e \Downarrow v$ instead of the notation $\sigma \vdash_q e \hookrightarrow v$ from [33]. Concretely, the pair of a program q and a substitution (i.e. partial function) σ constitutes a programming context $\langle q,\sigma\rangle$. Then, the expression judgement $\langle q,\sigma\rangle \vdash e \Downarrow v$ means that the expression e evaluates to the value v in the context $\langle q,\sigma\rangle$. Let us write $\langle q,\sigma\rangle \vdash e \Downarrow$ when there is some value v such that $\langle q,\sigma\rangle \vdash e \Downarrow v$.

As for the pattern matching operations which guide the formation of subtitutions, we replace $v \lhd l \rightsquigarrow \sigma$ [33, Fig. 3, pp.19] by the more restrictive statement $\langle q,\sigma\rangle \vdash l \Downarrow v$. The relation between those two kinds of expression is given by the following correspondance:

$$\frac{v \lhd l \rightsquigarrow \sigma}{\forall q.\, \langle q,\sigma\rangle \vdash e \Downarrow v}$$

This lead us to the following operational semantics which guarantees that computations are reversible, notably through the use of the instruction **rlet**.

$$\frac{}{\langle q,\{x\mapsto v\}\rangle \vdash x \Downarrow v} \qquad \frac{\lfloor v\rfloor \downarrow = v' \qquad \langle q,\sigma\rangle \vdash l \Downarrow v'}{\langle q,\sigma\rangle \vdash \lfloor l\rfloor \Downarrow v}$$

$$\frac{\langle q,\sigma_1\rangle \vdash l_1 \Downarrow v_1 \quad \cdots \quad \langle q,\sigma_n\rangle \vdash l_n \Downarrow v_n}{\langle q, \uplus_{i=1}^{n}\sigma_i\rangle \vdash c(l_1,\dots,l_n) \Downarrow c(v_1,\dots,v_n)} \qquad \frac{\begin{array}{c} f\, x_f \triangleq e_f \in q \quad \langle q,\sigma\rangle \vdash x \Downarrow v' \\ \langle q,\sigma_f\rangle \vdash x_f \Downarrow v' \quad \langle q,\sigma_f\rangle \vdash e_f \Downarrow v\end{array}}{\langle q,\sigma\rangle \vdash f\, x \Downarrow v}$$

$$\frac{\langle q, \sigma_{\text{in}} \rangle \vdash f\, l_{\text{in}} \Downarrow v_{\text{out}} \qquad \langle q, \sigma_{\text{out}} \uplus \sigma_e \rangle \vdash e \Downarrow v \qquad \langle q, \sigma_{\text{out}} \rangle \vdash l_{\text{out}} \Downarrow v_{\text{out}}}{\langle q, \sigma_{\text{in}} \uplus \sigma_e \rangle \vdash \textbf{let}\, l_{\text{out}} = f\, l_{\text{in}} \ \textbf{in}\ e \Downarrow v}$$

$$\frac{\langle q, \sigma_{\text{out}} \rangle \vdash f\, l_{\text{out}} \Downarrow v_{\text{in}} \qquad \langle q, \sigma_{\text{out}} \uplus \sigma_e \rangle \vdash e \Downarrow v \qquad \langle q, \sigma_{\text{in}} \rangle \vdash l_{\text{in}} \Downarrow v_{\text{in}}}{\langle q, \sigma_{\text{in}} \uplus \sigma_e \rangle \vdash \textbf{rlet}\, l_{\text{in}} = f\, l_{\text{out}} \ \textbf{in}\ e \Downarrow v}$$

$$\frac{\begin{array}{c}\langle q, \sigma_l \rangle \vdash l \Downarrow v' \qquad \langle q, \sigma_{l_j} \uplus \sigma_t \rangle \vdash e_j \Downarrow v \\ j = \min\{i \mid \forall q. \langle q, \sigma_{l_i} \rangle \vdash l_i \Downarrow v'\} \\ = \min\{i \mid \forall q.\, l' \in \text{leaves}(e_i) \wedge \langle q, - \rangle \vdash l' \Downarrow v\}\end{array}}{\sigma_l \uplus \sigma_t \vdash_q \textbf{case}\ l\ \textbf{of}\ \{l_i \to e_i\}_{i=1}^m \Downarrow v}$$

B.1 Denotational semantics

In this section, we provide a computationally adequate denotational semantics of Rfun [33] which relies on *bimonoidal* join inverse categories with decidable equality (and therefore decidable pattern matching). Like its operational semantics, the denotational semantics of Rfun depends on a given programming context.

Following the construction of the map lift in Sec. 3.3, the denotation of a variable x is taken in the substitution σ, *i.e.*

$$[\![x]\!]_q^\sigma = \text{lift}(s) \in T\mathcal{S} \qquad \text{with } s \overset{\text{def}}{=} \sigma(x)$$

Following the construction in Sec. 3.3, the denotation of a left expression $c(l_1, \ldots, l_n)$ (where $n \in \mathbb{N}$) is given by

$$[\![c(l_1, \ldots, l_n)]\!]_q^\sigma = \text{cons}_c([\![l_1]\!]_q^{\sigma_1}, \ldots, [\![l_n]\!]_q^{\sigma_n}) \in T\mathcal{S} \qquad \text{where } \sigma = \uplus_{i=1}^n \sigma_i$$

In particular, for tuples $\langle l_1, \ldots, l_n \rangle$, seen as a syntactic sugar for $\langle\rangle(l_1, \ldots, l_n)$:

$$[\![\langle l_1, \ldots, l_n \rangle]\!]_q^\sigma = \text{cons}_{\langle\rangle} \circ [\![(l_1, \ldots, l_n)]\!]_q^\sigma \in T\mathcal{S}$$

Exploiting Sec. 4.1, we associate the left expression $\lfloor l \rfloor$ with the following denotation,

$$\begin{aligned}[\![\lfloor l \rfloor]\!] &= \text{tag}_{\langle\rangle} \circ \iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})} \circ \text{dupeq}_{T\mathcal{S}} \circ [\![l]\!] \circ \text{dupeq}_{T\mathcal{S}}^\dagger \circ \iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})}^\dagger \circ \text{untag}_{\langle\rangle} \\ &: 1 \to T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})\end{aligned}$$

where $\iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})} = (\iota_1 \circ \amalg_1^\dagger) \vee (\iota_2 \circ \amalg_2^\dagger)$ with $\iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})}^\dagger$ complemented, since we required decidable equality.

Let-expressions and Rlet-expressions are given as follows, for every $f \in q$:

$$\begin{aligned}[\![\textbf{let}\ l_{\text{out}} = f(l_{\text{in}})\ \textbf{in}\ e]\!]_q^\sigma &= [\![e[f(l_{\text{in}})/l_{\text{out}}]]\!]_q^\sigma \\ [\![\textbf{rlet}\ l_{\text{in}} = f(l_{\text{out}})\ \textbf{in}\ e]\!]_q^\sigma &= [\![e[f^\dagger(l_{\text{in}})/l_{\text{out}}]]\!]_q^\sigma\end{aligned}$$

Case expressions are given their denotations via the presence of joins and decidable pattern matching. Recall that we have embeddings $\iota_n : T\mathcal{S}^{\otimes n} \to T\mathcal{S}$ with (complemented) projections $\iota_n^\dagger : T\mathcal{S} \to T\mathcal{S}^{\otimes n}$ from algebraic compactness and decidable pattern matching. We associate patterns p_i with restriction idempotents defined as follows:

$$\begin{aligned}
\{\!|x|\!\} &= \mathrm{id}_{T\mathcal{S}} \\
\{\!|c(p_1, \dots, p_n)|\!\} &= \mathrm{tag}_c \circ \iota_n \circ \{\!|p_1|\!\} \otimes \cdots \otimes \{\!|p_n|\!\} \circ \iota_n^\dagger \circ \mathrm{untag}_c \\
\{\!|\lfloor p \rfloor|\!\} &= \mathrm{tag}_{\langle\rangle} \circ \iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})} \circ \mathrm{dupeq}_{T\mathcal{S}} \circ \{\!|p|\!\} \circ \mathrm{dupeq}_{T\mathcal{S}}^\dagger \circ \iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})}^\dagger \circ \mathrm{untag}_{\langle\rangle}
\end{aligned}$$

where $\iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})} = (\iota_1 \circ \amalg_1^\dagger) \vee (\iota_2 \circ \amalg_2^\dagger)$.

Then, the denotation of case expressions can then be given by

$$[\![\textbf{case } l \textbf{ of } \{l_i \to e_i\}]\!]_q^\sigma = \left(\bigvee_{1 \leqslant i \leqslant n} [\![e_i]\!]_q^{\sigma_i} \circ \{\!|l_i|\!\} \circ \{\!|l_1|\!\}^\perp \circ \cdots \circ \{\!|l_{i-1}|\!\}^\perp \right) \circ [\![l]\!]_q^\sigma$$

where the substitution σ_i is generated as in the operational case.

Consider $(f x \triangleq e) \in q$. Then $[\![f x]\!]_q^\sigma : T\mathcal{S} \to T\mathcal{S}$ is defined by

$$\mathrm{fix}(r \mapsto (X \mapsto [\![e[x \mapsto X, f \mapsto r]]\!]_q^\sigma))$$

where fix is an instance of the fixpoint operator on hom-sets

$$\mathrm{fix} : (\mathrm{Hom}(T\mathcal{S}, T\mathcal{S}) \to \mathrm{Hom}(T\mathcal{S}, T\mathcal{S})) \to \mathrm{Hom}(T\mathcal{S}, T\mathcal{S})$$

constructed by means of the join (see [5, 18] for details). This operator corresponds to fixpoints over function identifiers, which is necessary to accomodate for the fact that the operational semantics allow for recursive calls.

Finally, a program $p \equiv d_1; \dots; d_n$ is denoted by

$$[\![p]\!]_q^\sigma = [\![d_1]\!]_q^\sigma \times \cdots \times [\![d_n]\!]_q^\sigma : T\mathcal{S}^{\otimes n} \to T\mathcal{S}^{\otimes n}$$

B.2 Computational adequacy

In the remaining part of this section, we show that bimonoidal join inverse categories with decidable equality form a class of computationally adequate models of the language Rfun.

Firstly, one can observe that every value is interpreted as a non-trivial total map and that the denotation is invariant up to a fixed context under our operational semantics, by straightforward induction.

▸ **Lemma 27.** *Every value v is interpreted as a non-trivial total map $[\![v]\!]_q^\sigma : 1 \to T\mathcal{S}$ in a bimonoidal join inverse category with decidable equality. Moreover, for every expression e such that the expression judgement $\langle q, \sigma \rangle \vdash e \Downarrow v$ holds for some substitution σ and some program q, the equality $[\![e]\!]_q^\sigma = [\![v]\!]_q^\sigma$ holds.*

Recall that we write $\langle q, \sigma \rangle \vdash e \Downarrow$ when there is some value v such that $\langle q, \sigma \rangle \vdash e \Downarrow v$. Then Lemma 27 leads to the following soundness theorem, as an intermediary step towards computational adequacy.

▸ **Theorem 28** (Soundness theorem). *In any bimonoidal join inverse category with decidable equality, for every expression e such that $\langle q, \sigma \rangle \vdash e \Downarrow$ holds for some programming context $\langle q, \sigma \rangle$, the denotational $[\![e]\!]_q^\sigma$ of the expression e is a non-trivial total map.*

Finally, we are required to prove that bimonoidal join inverse categories with decidable equality are computationally adequate models of the language Rfun. Recall that in such category, elements of an object X can be seen as maps $1 \to X$. In any bimonoidal join inverse category with dedicable equality $\mathbf{C}$, we write $[\![e]\!]^\sigma_q\!\downarrow$ when the denotation (taken in an object X of the category $\mathbf{C}$) of an expression e under the programming context $\langle q, \sigma \rangle$ is a non-trivial total map $1 \to X$.

▸ **Theorem 29** (Adequacy theorem). *In any bimonoidal join inverse category with decidable equality, for every expression e, the property $\langle q, \sigma \rangle \vdash e \Downarrow$ holds if and only if $[\![e]\!]^\sigma_q\!\downarrow$ holds.*

Proof. We already have one direction of the equivalence by Theorem 28. Therefore, to obtain computational adequacy, one needs to show that in any bimonoidal join inverse category with decidable equality, for every expression e under some context $\langle q, \sigma \rangle$, the property $[\![e]\!]^\sigma_q\!\downarrow$ implies that there is a value v such that $\langle q, \sigma \rangle \vdash e \Downarrow v$. Let's call it the adequacy property. This is straightforward when e is a variable x. For the rest, one can obtain the intended result by induction. In practice, we verify that $[\![e]\!]^\sigma_q$ is total by verifying that the equality $\mathrm{id}_1 = \overline{[\![e]\!]^\sigma_q}$ holds.

Consider a family $\{l_i\}_{1 \leqslant i \leqslant n}$ of left expressions under a family of contexts $\{\langle q, \sigma_i \rangle\}_{1 \leqslant i \leqslant n}$ (with $\sigma = \uplus_{i=1}^n \sigma_i$) which satisfy the adequacy property. Now we consider a symbol c and a left-expression $e \overset{\text{def}}{=} c(l_1, \ldots, l_n)$. In short, if the property $[\![e]\!]^\sigma_q\!\downarrow$ holds, then so does the property $[\![l_i]\!]^{\sigma_i}_q\!\downarrow$ for every i, since cons_c is a total map. In detail, keeping in mind that cons_c can be seen as a total map $T\mathcal{S}^{\otimes n} \to T\mathcal{S}$ (i.e. $\overline{\mathrm{cons}_c} = \mathrm{id}_{T\mathcal{S}^{\otimes n}}$) and that the projection maps of join inverse products are also total:

$$\begin{aligned} [\![e]\!]^\sigma_q\!\downarrow \implies [\![c(l_1, \ldots, l_n)]\!]^\sigma_q\!\downarrow &\implies \mathrm{id}_1 = \overline{\overline{\mathrm{cons}_c} \circ ([\![l_1]\!]^{\sigma_1}_q \otimes \cdots \otimes [\![l_n]\!]^{\sigma_n}_q)} \\ &\implies \mathrm{id}_1 = \bigotimes_{1 \leqslant i \leqslant n} \overline{[\![l_i]\!]^\sigma_q} \implies \forall 1 \leqslant i \leqslant n.\ \mathrm{id}_1 = \overline{[\![l_i]\!]^\sigma_1} \iff \forall 1 \leqslant i \leqslant n.\ [\![l_i]\!]^\sigma_1\!\downarrow \end{aligned}$$

Thus, for every i, by the adequacy property, there is a value v_i such that $\langle q, \sigma_i \rangle \vdash l_i \Downarrow v_i$ and therefore the value $v \overset{\text{def}}{=} c(v_1, \ldots, v_n)$ is such that $\langle q, \sigma \rangle \vdash e \Downarrow v$.

Similarly, consider the expression $e \overset{\text{def}}{=} \lfloor l \rfloor$ and assume that the property $[\![e]\!]^\sigma_q\!\downarrow$ holds and that l satisfies the adequacy property. Then the property $[\![l]\!]^\sigma_q\!\downarrow$ holds since $\mathrm{dupeq}_{T\mathcal{S}}$, $\mathrm{tag}_{\langle\rangle}$ and $\iota_{T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})}$ are total maps and $\mathrm{untag}_{\langle\rangle} = \mathrm{tag}^\dagger_{\langle\rangle}$. Indeed, the map

$$[\![e]\!]^\sigma_q = [\![\lfloor l \rfloor]\!]^\sigma_q : 1 \to T\mathcal{S} \oplus (T\mathcal{S} \otimes T\mathcal{S})$$

is a total map as the disjoint join of a total map $1 \to T\mathcal{S}$ and a total map $1 \to T\mathcal{S} \otimes T\mathcal{S}$. In technical terms, the denotation of the expression e can be rewritten as follows:

$$[\![e]\!] = \bigvee_{i \in \{1,2\}} \varphi_i \circ [\![l]\!] \circ \varphi_i^\dagger \qquad \text{where} \qquad \varphi_i \overset{\text{def}}{=} \mathrm{tag}_{\langle\rangle} \circ \iota_i \circ \amalg_i^\dagger \circ \mathrm{dupeq} \qquad \text{for } i \in \{1, 2\}$$

Therefore by the adequacy property, there is a value v such that $\langle q, \sigma \rangle \vdash l \Downarrow v$ and therefore the value $v' \overset{\text{def}}{=} \lfloor v \rfloor$ is such that $\langle q, \sigma \rangle \vdash e \Downarrow v'$.

Finally, by construction and straightforward induction, one obtains the adequacy property for case expressions and (r)let-expressions. ◂