Logical Methods in Computer Science
Vol. 14(3:16)2018, pp. 1–38
https://lmcs.episciences.org/

Submitted Oct. 12, 2017
Published Sep. 05, 2018

A CATEGORICAL FOUNDATION FOR STRUCTURED REVERSIBLE FLOWCHART LANGUAGES: SOUNDNESS AND ADEQUACY

ROBERT GLÜCK AND ROBIN KAARSGAARD

DIKU, Department of Computer Science, University of Copenhagen, Denmark
e-mail address: glueck@acm.org
e-mail address: robin@di.ku.dk

Abstract. Structured reversible flowchart languages is a class of imperative reversible programming languages allowing for a simple diagrammatic representation of control flow built from a limited set of control flow structures. This class includes the reversible programming language Janus (without recursion), as well as more recently developed reversible programming languages such as `R-CORE` and `R-WHILE`.

In the present paper, we develop a categorical foundation for this class of languages based on inverse categories with joins. We generalize the notion of extensivity of restriction categories to one that may be accommodated by inverse categories, and use the resulting *decisions* to give a reversible representation of predicates and assertions. This leads to a categorical semantics for structured reversible flowcharts, which we show to be computationally sound and adequate, as well as equationally fully abstract with respect to the operational semantics under certain conditions.

1. Introduction

Reversible computing is an emerging paradigm that adopts a physical principle of reality into a *computation model without information erasure*. Reversible computing extends the standard forward-only mode of computation with the ability to execute in reverse as easily as forward. Reversible computing is a necessity in the context of quantum computing and some bio-inspired computation models. Regardless of the physical motivation, bidirectional determinism is interesting in its own right. The potential benefits include the design of innovative reversible architectures (*e.g.*, [34, 33, 37]), new programming models and techniques (*e.g.*, [40, 17, 28]), and the enhancement of software with reversibility (*e.g.*, [8]).

Today the semantics of reversible programming languages are usually formalized using traditional metalanguages, such as structural operational semantics or denotational semantics

Key words and phrases: Reversible computing, flowchart languages, structured programming, categorical semantics, category theory.

This is the extended version of an article presented at MFPS XXXIII [16], extended with proofs that previously appeared in the appendix, as well as new sections on soundness, adequacy, and full abstraction.

The authors acknowledge the support given by *COST Action IC1405 Reversible computation: Extending horizons of computing.* We also thank the anonymous reviewers of MFPS XXXIII for their thoughtful and detailed comments on a previous version of this paper.

based on complete partial orders (*e.g.*, [19, 40]). However, these metalanguages were introduced for the definition of conventional (irreversible) programming languages. The fundamental properties of a reversible language, such as the required backward determinism and the invertibility of object language programs, are not naturally captured by these metalanguages. As a result, these properties are to be shown for each semantic definition. This unsatisfying state of affairs is not surprising, however, as these properties have been rarely required for more than half a century of mainstream language development. The recent advances in the area of reversible computing have changed this situation.

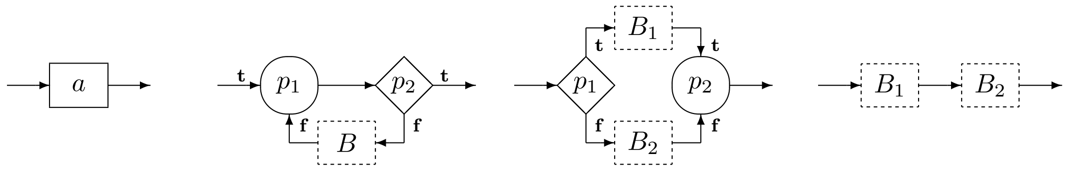
This paper provides a new categorical foundation specifically for formalizing reversible programming languages, in particular the semantics of reversible structured flowchart languages [36, 39], which are the reversible counterpart of the structured high-level programming languages used today. This formalization is based on join inverse categories with a developed notion of *extensivity* for inverse categories, which gives rise to natural representations of predicates and assertions, and consequently to models of reversible structured flowcharts. It provides a framework for modelling reversible languages, such that the reversible semantic properties of the object language are naturally ensured by the metalanguage.

The semantic framework we are going to present in this paper covers the reversible structured languages regardless of their concrete formation, such as atomic operations, elementary predicates, and value domains. State of the art reversible programming languages that are concretizations of this computation model include the well-known imperative language Janus [40] without recursion, the while languages `R-WHILE` and `R-CORE` with dynamic data structures [19, 20], and the structured reversible language SRL with stacks and arrays [39]. Structured control-flow is also a defining element of reversible object-oriented languages [22]. Further, unstructured reversible flowchart languages, such as reversible assembly languages with jumps [14, 4] and the unstructured reversible language RL [39], can be transformed into structured ones thanks to the structured reversible program theorem [36].

The main contribution of this paper is to provide a metalanguage formalism based on join inverse categories that is geared to formalize the reversible flowchart languages. The languages formalized in this framework are ensured to have the defining reversible language properties, including backward determinism and local invertibility. Another main property of the formalism is that every reversible structured language that is syntactically closed under inversion of its elementary operations is also closed under inversion of reversible control-flow operators. This is particularly useful, as it is sufficient to check this property for elementary constructs to ensure the correctness of the associated program inverter. Key to our formalism are decisions, which provide a particularly advantageous reversible representation of predicates and their corresponding assertions. This insight may guide the design of reversible constructs that are often quite involved to model in a reversible setting, such as pattern matching.

The results in this paper are illustrated by introducing a family of small reversible flowchart languages $\mathtt{RINT}_k$ for reversible computing with integer data, a reversible counterpart of the family of classic counter languages used for theoretical investigations into irreversible languages. The family introduced here may well serve the same purpose in a reversible context.

Overview: In Section 2, we give an introduction to structured reversible flowchart languages, while Section 3 describes the restriction and inverse category theory used as backdrop in later sections. In Section 4, we warm up by developing a notion of extensivity for inverse categories, based on extensive restriction categories and its associated concept



(a) Step (b) While loop (c) Selection (d) Sequence

Figure 1: Structured reversible flowcharts.

of *decisions*. Then, in Section 5, we put it all to use by showing how decisions may be used to model predicates and ultimately also reversible flowcharts, and we show that these are computationally sound and adequate with respect to the operational semantics in Section 6. In Section 7, we extend the previous theorems by giving a sufficient condition for equational full abstraction. In Section 8, we show how to verify program inversion using the categorical semantics, develop a small language to exemplify our framework, and discuss other applications in reversible programming. Section 9 offers some concluding remarks.

2. Reversible structured flowcharts

Structured reversible flowcharts [36, 39] naturally model the control flow behavior of reversible (imperative) programming languages in a simple diagrammatic representation, as classical flowcharts do for conventional languages. A crucial difference is that atomic steps are limited to *partial injective functions* and they require an additional *assertion*, an explicit orthogonalizing condition, at join points in the control flow.

A structured reversible flowchart F is built from four blocks (Figure 1): An *atomic step* that performs an elementary operation on a domain X specified by a partial injective function $a : X \rightharpoonup X$; a *while loop* over a block B with entry assertion $p_1 : X \to \mathit{Bool}$ and exit test $p_2 : X \to \mathit{Bool}$; a *selection* of block B_1 or B_2 with entry test $p_1 : X \to \mathit{Bool}$ and exit assertion $p_2 : X \to \mathit{Bool}$; and a *sequence* of blocks B_1 and B_2.

A structured reversible flowchart F consists of one main block. Blocks have unique entry and exit points, and can be nested any number of times to form more complex flowcharts. The interpretation of F consists of a given domain X (typically of stores or states, which we shall denote by σ) and a finite set of partial injective functions a and predicates $p : X \to \mathit{Bool}$. Computation starts at the entry point of F in an initial x_0 (the input), proceeds sequentially through the edges of F, and ends at the exit point of F in a final x_n (the output), if F is defined on the given input. Though the specific set of predicates depend on the flowchart language, they are often (as we will do here) assumed to be closed under Boolean operators, in particular conjunction and negation. The operational semantics for these, shown in Figure 2, are the same as in the irreversible case (see, *e.g.*, [35]): We use the judgment form $\sigma \vdash p \rightsquigarrow b$ here to mean that the predicate p evaluated on the state σ results in the Boolean value b.

The assertion p_1 in a reversible while loop (marked by a circle, as introduced in [40]) is a new flowchart operator: the predicate p_1 must be *true* when the control flow reaches the assertion along the **t**-edge, and *false* when it reaches the assertion along the **f**-edge; otherwise, the loop is undefined. The test p_2 (marked by a diamond) has the usual semantics. This means that B in a loop is repeated as long as p_1 and p_2 are *false*.

$$\boxed{\sigma \vdash p \rightsquigarrow b}$$

$$\frac{}{\sigma \vdash tt \rightsquigarrow tt} \qquad \frac{}{\sigma \vdash \mathit{ff} \rightsquigarrow \mathit{ff}}$$

$$\frac{\sigma \vdash p \rightsquigarrow tt}{\sigma \vdash \textbf{not } p \rightsquigarrow \mathit{ff}} \qquad \frac{\sigma \vdash p \rightsquigarrow \mathit{ff}}{\sigma \vdash \textbf{not } p \rightsquigarrow tt}$$

$$\frac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash q \rightsquigarrow tt}{\sigma \vdash p \textbf{ and } q \rightsquigarrow tt} \qquad \frac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash q \rightsquigarrow \mathit{ff}}{\sigma \vdash p \textbf{ and } q \rightsquigarrow \mathit{ff}}$$

$$\frac{\sigma \vdash p \rightsquigarrow \mathit{ff} \qquad \sigma \vdash q \rightsquigarrow tt}{\sigma \vdash p \textbf{ and } q \rightsquigarrow \mathit{ff}} \qquad \frac{\sigma \vdash p \rightsquigarrow \mathit{ff} \qquad \sigma \vdash q \rightsquigarrow \mathit{ff}}{\sigma \vdash p \textbf{ and } q \rightsquigarrow \mathit{ff}}$$

Figure 2: Operational semantics for Boolean predicates.

The selection has an assertion p_2, which must be *true* when the control flow reaches the assertion from B_1, and *false* when the control flow reaches the assertion from B_2; otherwise, the selection is undefined. As usual, the test p_1 selects B_1 or B_2. The assertion makes the selection reversible.

Despite their simplicity, reversible structured flowcharts are *reversibly universal* [3], which means that they are computationally as powerful as any reversible programming language can be. Given a suitable domain X for finite sets of atomic operations and predicates, there exists, for every injective computable function $f : X \to Y$, a reversible flowchart F that computes f.

Reversible structured flowcharts (Figure 1) have a straightforward representation as program texts defined by the grammar

$$B \quad ::= \quad a \quad | \quad \textbf{from } p \textbf{ loop } B \textbf{ until } p \quad | \quad \textbf{if } p \textbf{ then } B \textbf{ else } B \textbf{ fi } p \quad | \quad B \; ; B$$

It is often assumed, as we will do here, that the set of atomic steps contains a step **skip** that acts as the identity. The operational semantics for these flowchart structures (or simply *commands*) are shown in Figure 3. Here, the judgment form $\sigma \vdash c \downarrow \sigma'$ is used and taken to mean that the command c converges in the state σ resulting in a new state σ'. Note the use of the meta-command $\underline{loop}[p, c, q]$: This does not represent any reversible flowchart structure (and is thus not a syntactically valid command), but is rather a piece of internal syntax used to give meaning to loops. This is required since the role of the entry assertion in a reversible while loop changes after the first iteration: When entering the loop from the outside (*i.e.*, before any iterations have occured), the entry assertion must be true, but when entering from the inside (*i.e.*, one or more iterations), the entry assertion must be false.

The reversible structured flowcharts defined above corresponds to the reversible language `R-WHILE` [19], but their value domain, atomic functions and predicates are unspecified. As a minimum, a reversible flowchart needs blocks (a), (b), and (d) from Figure 1, because selection can be simulated by combining while loops that conditionally skip the body block or execute it once. `R-CORE` [20] is an example of such a minimal language.

3. Restriction and inverse categories

The following section contains the background on restriction and inverse category theory necessary for our later developments. Unless otherwise specified, the definitions and results

$$\boxed{\sigma \vdash c \downarrow \sigma'}$$

$$\frac{}{\sigma \vdash \textbf{skip} \downarrow \sigma} \qquad \frac{\sigma \vdash c_1 \downarrow \sigma' \qquad \sigma' \vdash c_2 \downarrow \sigma''}{\sigma \vdash c_1 \; ; \; c_2 \downarrow \sigma''}$$

$$\frac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash c_1 \downarrow \sigma' \qquad \sigma' \vdash q \rightsquigarrow tt}{\sigma \vdash \textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q \downarrow \sigma'} \qquad \frac{\sigma \vdash p \rightsquigarrow f\!f \qquad \sigma \vdash c_2 \downarrow \sigma' \qquad \sigma' \vdash q \rightsquigarrow f\!f}{\sigma \vdash \textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q \downarrow \sigma'}$$

$$\frac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash q \rightsquigarrow tt}{\sigma \vdash \textbf{from } p \textbf{ loop } c \textbf{ until } q \downarrow \sigma} \qquad \frac{\sigma \vdash p \rightsquigarrow f\!f \qquad \sigma \vdash q \rightsquigarrow tt}{\sigma \vdash \underline{loop}[p, c, q] \downarrow \sigma}$$

$$\frac{\sigma \vdash p \rightsquigarrow f\!f \qquad \sigma \vdash q \rightsquigarrow f\!f \qquad \sigma \vdash c \downarrow \sigma' \qquad \sigma' \vdash \underline{loop}[p, c, q] \downarrow \sigma''}{\sigma \vdash \underline{loop}[p, c, q] \downarrow \sigma''}$$

$$\frac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash q \rightsquigarrow f\!f \qquad \sigma \vdash c \downarrow \sigma' \qquad \sigma' \vdash \underline{loop}[p, c, q] \downarrow \sigma''}{\sigma \vdash \textbf{from } p \textbf{ loop } c \textbf{ until } q \downarrow \sigma''}$$

Figure 3: Operational semantics for the reversible flowchart structures.

presented in this section can be found in introductory texts on the subject (*e.g.*, [15, 21, 10, 11, 12]).

Restriction categories [10, 11, 12] axiomatize categories of partial maps. This is done by assigning to each morphism f a *restriction idempotent* $\overline{f}$, which we think of as a partial identity defined precisely where f is. Formally, restriction categories are defined as follows.

Definition 3.1. A *restriction category* is a category $\mathscr{C}$ equipped with a combinator mapping each morphism $A \xrightarrow{f} B$ to a morphism $A \xrightarrow{\overline{f}} A$ satisfying

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

for all suitable g.

As an example, the category **Pfn** of sets and partial functions is a restriction category, with $\overline{f}(x) = x$ if f is defined at x, and undefined otherwise. Note that being a restriction category is a structure, not a property; a category may be a restriction category in several different ways (*e.g.*, assigning $\overline{f} = \mathrm{id}$ for each morphism f gives a trivial restriction structure to any category).

In restriction categories, we say that a morphism $A \xrightarrow{f} B$ is *total* if $\overline{f} = \mathrm{id}_A$, and a *partial isomorphism* if there exists a (necessarily unique) *partial inverse* $B \xrightarrow{f^\dagger} A$ such that $f^\dagger f = \overline{f}$ and $ff^\dagger = \overline{f^\dagger}$. Isomorphisms are then simply the total partial isomorphisms with total partial inverses. An inverse category can then be defined as a special kind of restriction category[1].

Definition 3.2. An *inverse category* is a restriction category where each morphism is a partial isomorphism.

[1]This is a rather modern definition due to [10]. Originally, inverse categories were defined as the categorical extensions of inverse semigroups; see [27].

Every restriction category $\mathscr{C}$ gives rise to an inverse category $\mathrm{Inv}(\mathscr{C})$ (the *cofree* inverse category of $\mathscr{C}$, see [26]), which has as objects all objects of $\mathscr{C}$, and as morphisms all of the partial isomorphisms of $\mathscr{C}$. As such, since partial isomorphisms in **Pfn** are partial injective functions, a canonical example of an inverse category is the category $\mathrm{Inv}(\mathbf{Pfn}) \cong \mathbf{PInj}$ of sets and partial injective functions.

Since each morphism in an inverse category has a unique partial inverse, as also suggested by our notation this makes inverse categories canonically *dagger categories* [30], in the sense that they come equipped with a contravariant endofunctor $(-)^\dagger$ satisfying $f = f^{\dagger\dagger}$ and $\mathrm{id}_A^\dagger = \mathrm{id}_A$ for each morphism f and object A.

Given two restriction categories $\mathscr{C}$ and $\mathscr{D}$, the well-behaved functors between them are *restriction functors*, *i.e.*, functors F satisfying $F(\overline{f}) = \overline{F(f)}$. Analogous to how regular semigroup homomorphisms preserve partial inverses in inverse semigroups, when $\mathscr{C}$ and $\mathscr{D}$ are inverse categories, all functors between them are restriction functors; specifically they preserve the canonical dagger, *i.e.*, $F(f^\dagger) = F(f)^\dagger$.

Before we move on, we show some basic facts about restriction idempotents in restriction (and inverse) categories that will become useful later (see also, *e.g.*, [10, 15]).

Lemma 3.3. *In any restriction category, it is the case that*

(i) $\overline{id} = id$,
(ii) $\overline{\overline{f}} = \overline{f}$,
(iii) $\overline{gf}\ \overline{f} = \overline{gf}$,
(iv) $\overline{\overline{g}f} = \overline{gf}$,
(v) *if g is total then $\overline{gf} = \overline{f}$, and*
(vi) *if gf is total then so is f.*

for all morphisms $X \xrightarrow{f} Y$ and $Y \xrightarrow{g} Z$.

Proof. For (i) $\mathrm{id} = \mathrm{id}\ \overline{\mathrm{id}} = \overline{\mathrm{id}}$, and (ii) by $\overline{\overline{f}} = \overline{\mathrm{id}\overline{f}} = \overline{\mathrm{id}}\ \overline{f} = \mathrm{id}\overline{f} = \overline{f}$ using (i). For (iii) we have $\overline{gf}\ \overline{f} = \overline{gf\overline{f}} = \overline{gf}$, while (iv) follows by $\overline{\overline{g}f} = \overline{f\overline{gf}} = \overline{f}\ \overline{gf} = \overline{gf}\ \overline{f} = \overline{gf}$ using (iii). A special case of (iv) is (v) since g total means that $\overline{g} = \mathrm{id}$, and so $\overline{gf} = \overline{\overline{g}f} = \overline{\mathrm{id}f} = \overline{f}$. Finally, (vi) follows by gf total means that $\overline{gf} = \mathrm{id}$, so by this and (iii), $\mathrm{id} = \overline{gf} = \overline{gf}\ \overline{f} = \mathrm{id}\overline{f} = \overline{f}$, so f is total as well. □

3.1. **Partial order enrichment and joins.**

A consequence of how restriction (and inverse) categories are defined is that hom sets $\mathscr{C}(A,B)$ may be equipped with a partial order given by $f \le g$ iff $g\overline{f} = f$ (this extends to an enrichment in the category of partial orders and monotone functions). Intuitively, this states that f is below g iff g behaves exactly like f when restricted to the points where f is defined. Notice that any morphism e below an identity id_X is a restriction idempotent under this definition, since $e \le \mathrm{id}_X$ iff $\mathrm{id}_X\overline{e} = e$, *i.e.*, iff $\overline{e} = e$.

A sufficient condition for each $\mathscr{C}(A,B)$ to have a least element is that $\mathscr{C}$ has a *restriction zero*; a zero object 0 in the usual sense which additionally satisfies $A \xrightarrow{0_{A,A}} A = A \xrightarrow{\overline{0_{A,A}}} A$ for each endo-zero map $0_{A,A}$. One may now wonder when $\mathscr{C}(A,B)$ has joins as a partial order. Unfortunately, $\mathscr{C}(A,B)$ has joins of all morphisms only in very degenerate cases. However, if instead of considering arbitrary joins we consider joins of maps that are somehow compatible, this becomes much more viable.

Definition 3.4. In a restriction category, say that morphisms $X \xrightarrow{f} Y$ and $X \xrightarrow{g} Y$ are *disjoint* iff $f\overline{g} = 0$; and *compatible* iff $f\overline{g} = g\overline{f}$.

It can be shown that disjointness implies compatibility, as disjointness is expectedly symmetric. Further, we may extend this to say that a set of parallel morphisms is disjoint iff each pair of morphisms is disjoint, and likewise for compatibility. This gives suitable notions of *join restriction categories.*

Definition 3.5. A restriction category $\mathscr{C}$ has compatible (disjoint) joins if it has a restriction zero, and satisfies that for each compatible (disjoint) subset S of any hom set $\mathscr{C}(A, B)$, there exists a morphism $\bigvee_{s\in S} s$ such that

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

For inverse categories, the situation is a bit more tricky, as the join of two compatible partial isomorphisms may not be a partial isomorphism. To ensure this, we need stronger relations:

Definition 3.6. In an inverse category, say that parallel maps f and g are *disjoint* iff $f\overline{g} = 0$ and $f^\dagger\overline{g^\dagger} = 0$; and *compatible* iff $f\overline{g} = g\overline{f}$ and $f^\dagger\overline{g^\dagger} = g^\dagger\overline{f^\dagger}$.

We may now extend this to notions of disjoint sets and compatible sets of morphisms in inverse categories as before. This finally gives notions of *join inverse categories*:

Definition 3.7. An inverse category $\mathscr{C}$ has compatible (disjoint) joins if it has a restriction zero and satisfies that for all compatible (disjoint) subsets S of all hom sets $\mathscr{C}(A, B)$, there exists a morphism $\vee_{s\in S} s$ satisfying (i) – (iv) of Definition 3.5.

An example of a join inverse category is **PInj** of sets and partial injective functions. Here, $\bigvee_{f\in S} f$ for a set of compatible partial injective functions S is constructed as the function with graph the union of the graphs of all $f \in S$; or equivalently as the function

$$\left(\bigvee_{f\in S} f\right)(x) = \begin{cases} g(x) & \text{if there exists } g \in S \text{ such that } g(x) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Similarly, the category **PHom** of topological spaces and partial homeomorphisms (*i.e.*, partial injective functions which are open and continuous in their domain of definition) is an inverse category with joins constructed in the same way. Further, for any join restriction category $\mathscr{C}$, the cofree inverse category $\mathrm{Inv}(\mathscr{C})$ of $\mathscr{C}$ is a join inverse category as well [21, Lemma 3.1.27], and joins on both restriction and inverse categories can be freely adjoined [21, Sec. 3.1]. A functor F between restriction (or inverse) categories with joins is said to be join-preserving when $F(\bigvee_{s\in S} s) = \bigvee_{s\in S} F(s)$.

3.2. **Restriction coproducts, extensivity, and related concepts.** While a restriction category may very well have coproducts, these are ultimately only well-behaved when the coproduct injections $A \xrightarrow{\kappa_1} A + B$ and $B \xrightarrow{\kappa_2} A + B$ are total; if this is the case, we say that the restriction category has *restriction coproducts.* If a restriction category has all finite restriction coproducts, it also has a restriction zero serving as unit. Note that in restriction categories with restriction coproducts, the coproduct injections κ_1 (respectively κ_2) are partial isomorphisms with partial inverse $\kappa_1^\dagger = A + B \xrightarrow{[\mathrm{id}_A, 0_{B,A}]} A$ (respectively $\kappa_2^\dagger = A + B \xrightarrow{[0_{A,B}, \mathrm{id}_B]} A$).

In [12], it is shown that the existence of certain maps, called *decisions*, in a restriction category $\mathscr{C}$ with restriction coproducts leads to the subcategory Total($\mathscr{C}$) of total maps being extensive (in the sense of, *e.g.*, [7]). This leads to the definition of an *extensive restriction category*[2].

Definition 3.8. A restriction category is said to be *extensive* (as a restriction category) if it has restriction coproducts and a restriction zero, and for each map $A \xrightarrow{f} B + C$ there is a unique *decision* $A \xrightarrow{\langle f \rangle} A + A$ satisfying

(D.1): $\nabla \langle f \rangle = \overline{f}$ and **(D.2):** $(f + f)\langle f \rangle = (\kappa_1 + \kappa_2) f$.

In the above, ∇ denotes the codiagonal $[\mathrm{id}, \mathrm{id}]$. A consequence of these axioms is that each decision is a partial isomorphism; one can show that $\langle f \rangle$ must be partial inverse to $[\overline{\kappa_1^\dagger f}, \overline{\kappa_2^\dagger f}]$ (see [12]). Further, when a restriction category with restriction coproducts has finite joins, it is also extensive with $\langle f \rangle = (\kappa_1 \overline{\kappa_1^\dagger f}) \vee (\kappa_2 \overline{\kappa_2^\dagger f})$. As an example, **Pfn** is extensive with $A \xrightarrow{\langle f \rangle} A + A$ for $A \xrightarrow{f} B + C$ given by

$$\langle f \rangle(x) = \begin{cases} \kappa_1(x) & \text{if } f(x) = \kappa_1(y) \text{ for some } y \in B \\ \kappa_2(x) & \text{if } f(x) = \kappa_2(z) \text{ for some } z \in C \\ \text{undefined} & \text{if } f(x) \text{ is undefined} \end{cases} .$$

While inverse categories only have coproducts (much less restriction coproducts) in very degenerate cases (see [15]), they may very well be equipped with a more general sum-like symmetric monoidal tensor, a disjointness tensor.

Definition 3.9. A *disjointness tensor* on a restriction category is a symmetric monoidal restriction functor $- \oplus -$ satisfying that its unit is the restriction zero, and that the canonical maps

$$\amalg_1 = A \xrightarrow{\rho^{-1}} A \oplus 0 \xrightarrow{\mathrm{id} \oplus 0} A \oplus B \qquad \amalg_2 = B \xrightarrow{\lambda^{-1}} 0 \oplus B \xrightarrow{0 \oplus \mathrm{id}} A \oplus B$$

are jointly epic, where ρ respectively λ is the left respectively right unitor of the monoidal functor $- \oplus -$.

It can be straightforwardly shown that any restriction coproduct gives rise to a disjointness tensor. A useful interaction between compatible joins and a join-preserving disjointness tensor in inverse categories was shown in [5, 26], namely that it leads to a †-trace (in the sense of [24, 31]):

Proposition 3.10. *Let $\mathscr{C}$ be an inverse category with (at least countable) compatible joins and a join-preserving disjointness tensor. Then $\mathscr{C}$ has a trace operator given by*

$$\mathrm{Tr}_{A,B}^{U}(f) = f_{11} \vee \bigvee_{n \in \omega} f_{21} f_{22}^{n} f_{12}$$

[2]The name is admittedly mildly confusing, as an extensive restriction category is not extensive in the usual sense. Nevertheless, we stay with the established terminology.

satisfying $\mathrm{Tr}^U_{A,B}(f)^\dagger = \mathrm{Tr}^U_{A,B}(f^\dagger)$*, where* $f_{ij} = \amalg_j^\dagger f \amalg_i$.

4. Extensivity of inverse categories

As discussed earlier, extensivity of restriction categories hinges on the existence of certain partial isomorphisms – decisions – yet their axiomatization relies on the presence of a map that is not a partial isomorphism, the codiagonal.

In this section, we tweak the axiomatization of extensivity of restriction categories to one that is equivalent, but additionally transports more easily to inverse categories. We then give a definition of extensitivity for inverse categories, from which it follows that $\mathrm{Inv}(\mathscr{C})$ is an extensive inverse category when $\mathscr{C}$ is an extensive restriction category.

Recall that decisions satisfy the following two axioms:

(D.1): $\nabla\langle f\rangle = \overline{f}$ and **(D.2):** $(f+f)\langle f\rangle = (\kappa_1+\kappa_2)f$

As mentioned previously, an immediate problem with this is the reliance on the codiagonal. However, intuitively, what **(D.1)** states is simply that the decision $\langle f\rangle$ cannot do anything besides to tag its inputs appropriately. Using a disjoint join, we reformulate this axiom to the following:

(D'.1): $(\kappa_1^\dagger\langle f\rangle) \vee (\kappa_2^\dagger\langle f\rangle) = \overline{f}$

Note that this axiom also subtly states that disjoint joins of the given form always exist.

Say that a restriction category is *pre-extensive* if it has restriction coproducts, a restriction zero, and a combinator mapping each map $A \xrightarrow{f} B+C$ to a *pre-decision* $A \xrightarrow{\langle f\rangle} A+A$ (with no additional requirements). We can then show the following:

Theorem 4.1. *Let $\mathscr{C}$ be a pre-extensive restriction category. The following are equivalent:*
(i) *$\mathscr{C}$ is an extensive restriction category.*
(ii) *Every pre-decision of $\mathscr{C}$ satisfies **(D.1)** and **(D.2)**.*
(iii) *Every pre-decision of $\mathscr{C}$ satisfies **(D'.1)** and **(D.2)**.*

To show this theorem, we will need the following lemma:

Lemma 4.2. *In an extensive restriction category, joins of the form $(f+0)\vee(0+g)$ exist for all maps $A \xrightarrow{f} B$ and $C \xrightarrow{g} D$ and are equal to $f+g$.*

Proof. By [9], for any map $A \xrightarrow{h} B+C$ in an extensive restriction category, $h = (\overline{\kappa_1^\dagger h}) \vee (\overline{\kappa_2^\dagger h})$. But then $f+g = (\overline{\kappa_1^\dagger(f+g)}) \vee (\overline{\kappa_2^\dagger(f+g)}) = ((\mathrm{id}+0)(f+g)) \vee ((0+\mathrm{id})(f+g)) = (f+0)\vee(0+g)$. □

We can now continue with the proof.

Proof. The equivalence between (i) and (ii) was given in [12]. That (ii) and (iii) are equivalent follows by

$$\begin{aligned}(\kappa_1^\dagger\langle f\rangle) \vee (\kappa_2^\dagger\langle f\rangle) &= ([\mathrm{id},0]\langle f\rangle) \vee ([0,\mathrm{id}]\langle f\rangle)\\ &= (\nabla(\mathrm{id}+0)\langle f\rangle) \vee (\nabla(0+\mathrm{id})\langle f\rangle)\\ &= \nabla((\mathrm{id}+0)\vee(0+\mathrm{id}))\langle f\rangle\\ &= \nabla(\mathrm{id}+\mathrm{id})\langle f\rangle = \nabla\langle f\rangle\end{aligned}$$

where we note that the join $(\mathrm{id}+0) \vee (0+\mathrm{id})$ exists and equals $\mathrm{id}+\mathrm{id}$ when every pre-decision satisfies **(D.1)** and **(D.2)** by Lemma 4.2. That the join also exists when every pre-decision satisfies **(D'.1)** and **(D.2)** follows as well, since the universal mapping property for coproducts guarantees that the only map g satisfying $((\kappa_1+\kappa_2)+(\kappa_1+\kappa_2))g = (\kappa_1+\kappa_2)(\kappa_1+\kappa_2)$ is $\kappa_1+\kappa_2$ itself, so we must have $\langle \kappa_1+\kappa_2 \rangle = \kappa_1+\kappa_2$, and

$$\begin{aligned}(\mathrm{id}+0) \vee (0+\mathrm{id}) &= \overline{\kappa_1^\dagger} \vee \overline{\kappa_2^\dagger} = (\kappa_1\kappa_1^\dagger) \vee (\kappa_2\kappa_2^\dagger) \\ &= (\kappa_1^\dagger(\kappa_1+\kappa_2)) \vee (\kappa_2^\dagger(\kappa_1+\kappa_2)) \\ &= \overline{\kappa_1+\kappa_2} = \mathrm{id}+\mathrm{id}\end{aligned}$$

which was what we wanted. □

Another subtle consequence of our amended first rule is that $\kappa_1^\dagger\langle f\rangle$ is its own restriction idempotent (and likewise for $\kappa_2^\dagger\langle f\rangle$) since $\kappa_1^\dagger\langle f\rangle \le (\kappa_1^\dagger\langle f\rangle) \vee (\kappa_2^\dagger\langle f\rangle) = \overline{f} \le \mathrm{id}$, as the maps below identity are precisely the restriction idempotents.

Our next snag in transporting this definition to inverse categories has to do with the restriction coproducts themselves, as it is observed in [15] that any inverse category with restriction coproducts is a preorder. Intuitively, the problem is not that unicity of coproduct maps cannot be guaranteed in non-preorder inverse categories, but rather that the coproduct map $A+B \xrightarrow{[f,g]} C$ in a restriction category is not guaranteed to be a partial isomorphism when f and g are.

For this reason, we will consider the more general disjointness tensor for sum-like constructions rather than full-on restriction coproducts, as inverse categories may very well have a disjointness tensor without it leading to immediate degeneracy. Notably, **PInj** has a disjointness tensor, constructed on objects as the disjoint union of sets (precisely as the restriction coproduct in **Pfn**, but without the requirement of a universal mapping property). This leads us to the following definition:

Definition 4.3. An inverse category with a disjointness tensor is said to be *extensive* when each map $A \xrightarrow{f} B \oplus C$ has a unique decision $A \xrightarrow{\langle f\rangle} A \oplus A$ satisfying

(D'.1): $(\amalg_1^\dagger\langle f\rangle) \vee (\amalg_2^\dagger\langle f\rangle) = \overline{f}$

(D'.2): $(f \oplus f)\langle f\rangle = (\amalg_1 \oplus \amalg_2)f$.

As an example, **PInj** is an extensive inverse category with the unique decision $A \xrightarrow{\langle f\rangle} A \oplus A$ for a partial injection $A \xrightarrow{f} B \oplus C$ given by

$$\langle f\rangle(x) = \begin{cases} \amalg_1(x) & \text{if } f(x) = \amalg_1(y) \text{ for some } y \in B \\ \amalg_2(x) & \text{if } f(x) = \amalg_2(z) \text{ for some } z \in C \\ \text{undefined} & \text{if } f(x) \text{ is undefined} \end{cases} .$$

Aside from a shift from coproduct injections to the quasi-injections of the disjointness tensor, a subtle change here is the notion of join. That is, for restriction categories with disjoint joins, any pair of maps f, g with $f\overline{g} = 0$ has a join – but for inverse categories, we additionally require that their *inverses* are disjoint as well, *i.e.*, that $f^\dagger\overline{g^\dagger} = 0$, for the join to exist. In this case, however, there is no difference between the two. As previously discussed, a direct consequence of this axiom is that each $\amalg_i^\dagger\langle f\rangle$ must be its own restriction idempotent. Since restriction idempotents are self-adjoint (*i.e.*, satisfy $f = f^\dagger$), they are disjoint iff their inverses are disjoint.

Since restriction coproducts give rise to a disjointness tensor, we may straightforwardly show the following theorem.

Theorem 4.4. *When $\mathscr{C}$ is an extensive restriction category,* $\mathrm{Inv}(\mathscr{C})$ *is an extensive inverse category.*

Further, constructing the decision $\langle f\rangle$ as $(\amalg_1\overline{\amalg_1^\dagger f}) \vee (\amalg_2\overline{\amalg_2^\dagger f})$ (*i.e.*, mirroring the construction of decisions in restriction categories with disjoint joins), we may show the following.

Theorem 4.5. *Let $\mathscr{C}$ be an inverse category with a disjointness tensor, a restriction zero, and finite disjoint joins which are further preserved by the disjointness tensor. Then $\mathscr{C}$ is extensive as an inverse category.*

Before we proceed, we show two small lemmas regarding decisions and the (join preserving) disjointness tensor (for the latter, see also [15]).

Lemma 4.6. *In any inverse category with a disjointness tensor, it is the case that*
(i) $\overline{\amalg_1} = \overline{\amalg_2} = id$,
(ii) $\overline{\amalg_1^\dagger} = id \oplus 0$ *and* $\overline{\amalg_2^\dagger} = 0 \oplus id$,
(iii) $\amalg_j^\dagger \amalg_i = id$ *if* $i = j$ *and* $\amalg_j^\dagger \amalg_i = 0$ *otherwise.*

Proof. For (i), adding subscripts on identities and zero maps for clarity,

$$\begin{aligned}\overline{\amalg_1} &= \overline{(\mathrm{id}_X \oplus 0_{0,Y})\rho^{-1}} = \overline{\overline{(\mathrm{id}_X \oplus 0_{0,Y})}\rho^{-1}} = \overline{(\overline{\mathrm{id}_X} \oplus \overline{0_{0,Y}})\rho^{-1}} = \overline{(\mathrm{id}_X \oplus 0_{0,0})\rho^{-1}}\\ &= \overline{(\mathrm{id}_X \oplus \mathrm{id}_0)\rho^{-1}} = \overline{\mathrm{id}_{X\oplus 0}\ \rho^{-1}} = \overline{\rho^{-1}} = \mathrm{id}_X\end{aligned}$$

and analogously for $\overline{\amalg_2}$. For (ii),

$$\overline{\amalg_1^\dagger} = \overline{((\mathrm{id}\oplus 0)\rho^{-1})^\dagger} = \overline{\rho(\mathrm{id}^\dagger \oplus 0^\dagger)} = \overline{\overline{\rho}(\mathrm{id}\oplus 0)} = \overline{\mathrm{id}\ (\mathrm{id}\oplus 0)} = \overline{\mathrm{id}\oplus 0} = \mathrm{id}\oplus 0$$

and again, the proof for $\amalg_2$ is analogous. To show (iii) when $i = j$

$$\begin{aligned}\amalg_1^\dagger\amalg_1 &= \rho(\mathrm{id}_X \oplus 0_{Y,0})(\mathrm{id}_X \oplus 0_{0,Y})\rho^{-1} = \rho(\mathrm{id}_X \oplus 0_{0,0})\rho^{-1} = \rho(\mathrm{id}_X \oplus \mathrm{id}_0)\rho^{-1}\\ &= \rho\ \mathrm{id}_{X\oplus 0}\ \rho^{-1} = \rho\rho^{-1} = \mathrm{id}_X\end{aligned}$$

and similarly for $\amalg_2^\dagger\amalg_2 = \mathrm{id}$ (noting that $0_{Y,0}0_{0,Y} = 0_{0,0} = \mathrm{id}_0$ follows by the universal mapping property of the zero object 0). For $i \neq j$, we proceed with the case where $i = 1$ and $j = 2$, yielding

$$\amalg_2^\dagger\amalg_1 = \rho(0_{X,0} \oplus \mathrm{id}_Y)(\mathrm{id}_X \oplus 0_{0,Y})\rho^{-1} = \rho(0_{X,0} \oplus 0_{0,Y})\rho^{-1} = \rho\ 0_{X\oplus 0,0\oplus Y}\ \rho^{-1} = 0_{X,Y}$$

where $0_{X,0} \oplus 0_{0,Y} = 0_{X\oplus 0,0\oplus Y}$ follows by the fact that $0_{X,0} \oplus 0_{0,Y}$ factors through $0 \oplus 0$ (by the universal mapping property of 0) and the fact that $0 \oplus 0 \cong 0$ (since 0 serves as unit for $- \oplus -$), so $0_{X,0} \oplus 0_{0,Y}$ factors through 0 as well. The case where $i = 2$ and $j = 1$ is entirely analogous. □

Lemma 4.7. *In any extensive inverse category, it is the case that*
(i) $\overline{\amalg_1^\dagger\langle f\rangle} = \amalg_1^\dagger\langle f\rangle$,
(ii) $\overline{\amalg_1^\dagger} \vee \overline{\amalg_2^\dagger} = id$,
(iii) $\langle f\rangle = (\amalg_1\overline{\amalg_1^\dagger\langle f\rangle}) \vee (\amalg_2\overline{\amalg_2^\dagger\langle f\rangle})$, *and*
(iv) $\overline{f} = \overline{\langle f\rangle}$.

Proof. To show (i), we have by definition of the join that $\amalg_1^\dagger\langle f\rangle \leq (\amalg_1^\dagger\langle f\rangle) \vee (\amalg_2^\dagger\langle f\rangle) = \overline{f}$, where the equality follows precisely by the first axiom of decisions. Since $\overline{f} \leq \mathrm{id}$ by $\mathrm{id}\overline{\overline{f}} = \overline{f}$ (since $\overline{\overline{f}} = \overline{f}$ by Lemma 3.3), it follows by transitivity of $- \leq -$ (since it is a partial order) that $\amalg_1^\dagger\langle f\rangle \leq \mathrm{id}$. But since $f \leq g$ iff $g\overline{f} = f$, it follows from $\amalg_1^\dagger\langle f\rangle \leq \mathrm{id}$ that $\overline{\amalg_1^\dagger\langle f\rangle} = \mathrm{id}\,\overline{\amalg_1^\dagger\langle f\rangle} = \amalg_1^\dagger\langle f\rangle$.

For (ii), using Lemma 4.6 we start by noting that

$$\overline{\amalg_1^\dagger(\amalg_1 \oplus \amalg_2)} = \overline{\overline{\amalg_1^\dagger}(\amalg_1 \oplus \amalg_2)} = \overline{(\mathrm{id} \oplus 0)(\amalg_1 \oplus \amalg_2)} = \overline{\amalg_1 \oplus 0} = \overline{\amalg_1} \oplus \overline{0} = \mathrm{id} \oplus 0 = \overline{\amalg_1^\dagger}$$

and similarly

$$\overline{\amalg_2^\dagger(\amalg_1 \oplus \amalg_2)} = \overline{\overline{\amalg_2^\dagger}(\amalg_1 \oplus \amalg_2)} = \overline{(0 \oplus \mathrm{id})(\amalg_1 \oplus \amalg_2)} = \overline{0 \oplus \amalg_2} = \overline{0} \oplus \overline{\amalg_2} = 0 \oplus \mathrm{id} = \overline{\amalg_2^\dagger}$$

and since $\langle \amalg_1 \oplus \amalg_2 \rangle = \amalg_1 \oplus \amalg_2$, we have

$$\begin{aligned} \overline{\amalg_1^\dagger} \vee \overline{\amalg_2^\dagger} &= \overline{\amalg_1^\dagger(\amalg_1 \oplus \amalg_2)} \vee \overline{\amalg_2^\dagger(\amalg_1 \oplus \amalg_2)} = \overline{\amalg_1^\dagger\langle \amalg_1 \oplus \amalg_2 \rangle} \vee \overline{\amalg_2^\dagger\langle \amalg_1 \oplus \amalg_2 \rangle} \\ &= \overline{\overline{\amalg_1 \oplus \amalg_2}} = \overline{\amalg_1 \oplus \amalg_2} = \overline{\amalg_1} \oplus \overline{\amalg_2} = \mathrm{id} \oplus \mathrm{id} = \mathrm{id}. \end{aligned}$$

Using this, we may prove (iii) (and later (iv)) as follows:

$$\begin{aligned} \langle f \rangle &= \mathrm{id}\langle f \rangle = (\overline{\amalg_1^\dagger} \vee \overline{\amalg_2^\dagger})\langle f \rangle = (\overline{\amalg_1^\dagger}\langle f \rangle) \vee (\overline{\amalg_2^\dagger}\langle f \rangle) = (\amalg_1 \amalg_1^\dagger\langle f \rangle) \vee (\amalg_2 \amalg_2^\dagger\langle f \rangle) \\ &= (\amalg_1 \overline{\amalg_1^\dagger\langle f \rangle}) \vee (\amalg_2 \overline{\amalg_2^\dagger\langle f \rangle}) \end{aligned}$$

where $\amalg_1^\dagger\langle f \rangle = \overline{\amalg_1^\dagger\langle f \rangle}$ follows by (i). Using these again, we show (iv) by

$$\begin{aligned} \overline{f} &= (\amalg_1^\dagger\langle f \rangle) \vee (\amalg_2^\dagger\langle f \rangle) = \overline{\amalg_1^\dagger\langle f \rangle} \vee \overline{\amalg_2^\dagger\langle f \rangle} = \overline{\overline{\amalg_1^\dagger}\langle f \rangle} \vee \overline{\overline{\amalg_2^\dagger}\langle f \rangle} = \overline{(\overline{\amalg_1^\dagger}\langle f \rangle) \vee (\overline{\amalg_2^\dagger}\langle f \rangle)} \\ &= \overline{(\overline{\amalg_1^\dagger} \vee \overline{\amalg_2^\dagger})\langle f \rangle)} = \overline{\mathrm{id}\langle f \rangle} = \overline{\langle f \rangle} \end{aligned}$$

which was what we wanted. ∎

5. Modelling structured reversible flowcharts

In the following, let $\mathscr{C}$ be an inverse category with (at least countable) compatible joins and a join-preserving disjointness tensor. As disjoint joins are compatible, it follows that $\mathscr{C}$ is an extensive inverse category with a (uniform) †-trace operator.

In this section, we will show how this framework can be used model reversible structured flowchart languages. First, we will show how decisions in extensive inverse categories can be used to model predicates, and then how this representation extends to give very natural semantics to reversible flowcharts corresponding to conditionals and loops.

5.1. **Predicates as decisions.** In suitably equipped categories, one naturally considers predicates on an object A as given by maps $A \to 1 + 1$. In inverse categories, however, the mere idea of a predicate as a map of the form $A \to 1 \oplus 1$ is problematic, as only very degenerate maps of this form are partial isomorphisms. In the following, we show how decisions give rise to an unconventional yet ultimately useful representation of predicates. To our knowledge this representation is novel, motivated here by the necessity to model predicates in a reversible fashion, as decisions are always partial isomorphisms.

The simplest useful predicates are the predicates that are always true (respectively always false). By convention, we represent these by the left (respectively right) injection (which are both their own decisions),

$$\begin{aligned} [\![tt]\!] &= \amalg_1 \\ [\![ff]\!] &= \amalg_2 . \end{aligned}$$

Semantically, we may think of decisions as a separation of an object A into *witnesses* and *counterexamples* of the predicate it represents. In a certain sense, the axioms of decisions say that there is nothing more to a decision than how it behaves when postcomposed with $\amalg_1^\dagger$ or $\amalg_2^\dagger$. As such, given the convention above, we think of $\amalg_1^\dagger \langle p \rangle$ as the witnesses of the predicate represented by the decision $\langle p \rangle$, and $\amalg_2^\dagger \langle p \rangle$ as its counterexamples.

With this in mind, we turn to boolean combinators. The negation of a predicate-as-a-decision must simply swap witnesses for counterexamples (and vice versa). In other words, we obtain the negation of a decision by postcomposing with the commutator γ of the disjointness tensor,

$$[\![\textbf{not } p]\!] = \gamma \, [\![p]\!] .$$

With this, it is straightforward to verify that, *e.g.*, $[\![\textbf{not } tt]\!] = [\![ff]\!]$, as

$$[\![\textbf{not } tt]\!] = \gamma \amalg_1 = \gamma (\mathrm{id} \oplus 0) \rho^{-1} = (0 \oplus \mathrm{id}) \gamma \rho^{-1} = (0 \oplus \mathrm{id}) \lambda^{-1} = \amalg_2 = [\![ff]\!] .$$

For conjunction, we exploit that our category has (specifically) finite disjoint joins, and define the conjunction of predicates-as-decisions $[\![p]\!]$ and $[\![q]\!]$ by

$$[\![p \textbf{ and } q]\!] = \Big((\amalg_1 \overline{\amalg_1^\dagger \, [\![p]\!]} \; \overline{\amalg_1^\dagger \, [\![q]\!]}) \vee (\amalg_2 (\overline{\amalg_2^\dagger \, [\![p]\!]} \vee \overline{\amalg_2^\dagger \, [\![q]\!]})) \Big) \, \overline{[\![p]\!]} \; \overline{[\![q]\!]} .$$

The intuition behind this definition is that the witnesses of a conjunction of predicates is given by the meet of the witnesses of the each predicate, while the counterexamples of a conjunction of predicates is the join of the counterexamples of each predicate. Note that this is then precomposed with $\overline{[\![p]\!]} \; \overline{[\![q]\!]}$ to ensure that the result is only defined where both p and q are; this gives

Noting that the meet of two restriction idempotents is given by their composition, this is precisely what this definition states. Similarly we define the disjunction of $[\![p]\!]$ and $[\![q]\!]$ by

$$[\![p \textbf{ or } q]\!] = \Big((\amalg_1 (\overline{\amalg_1^\dagger \, [\![p]\!]} \vee \overline{\amalg_1^\dagger \, [\![q]\!]})) \vee (\amalg_2 \overline{\amalg_2^\dagger \, [\![p]\!]} \; \overline{\amalg_2^\dagger \, [\![q]\!]}) \Big) \, \overline{[\![p]\!]} \; \overline{[\![q]\!]} ,$$

as $[\![p \textbf{ or } q]\!]$ then has as witnesses the join of the witnesses of $[\![p]\!]$ and $[\![q]\!]$, and as counterexamples the meet of the counterexamples of $[\![p]\!]$ and $[\![q]\!]$. With these definitions, it can be shown that, *e.g.*, the De Morgan laws are satisfied. However, since we can thus construct this from conjunctions and negations, we will leave disjunctions as syntactic sugar.

That all of these are indeed decisions can be shown straightforwardly, as summarized in the following closure theorem.

Theorem 5.1. *Decisions in $\mathscr{C}$ are closed under Boolean negation, conjunction, and disjunction.*

5.2. **Reversible structured flowcharts, categorically.** To give a categorical account of structured reversible flowchart languages, we assume the existence of a suitable distinguished object Σ of stores, which we think of as the *domain of computation*, such that we may give denotations to structured reversible flowcharts as morphisms $\Sigma \to \Sigma$.

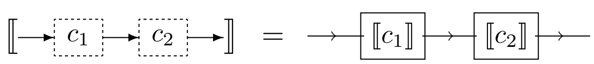
Since atomic steps (corresponding to elementary operations, *e.g.*, store updates) may vary from language to language, we assume that each such atomic step in our language has a denotation as a morphism $\Sigma \to \Sigma$. In the realm of reversible flowcharts, these atomic steps are required to be partial injective functions; here, we abstract this to require that their denotation is a partial isomorphism (though this is a trivial requirement in inverse categories).

Likewise, elementary predicates (*e.g.*, comparison of values in a store) may vary from language to language, so we assume that such elementary predicates have denotations as well as decisions $\Sigma \to \Sigma \oplus \Sigma$. If necessary (as is the case for Janus [40]), we may then close these elementary predicates under boolean combinations as discussed in the previous section.

To start, we note how sequencing of flowcharts may be modelled trivially by means of composition, *i.e.*,

$$[\![c_1 \ ; c_2]\!] = [\![c_2]\!]\,[\![c_1]\!]$$

or, using the diagrammatic notation of flowcharts and the string diagrams for monoidal categories in the style of [31] (read left-to-right and bottom-to-top),

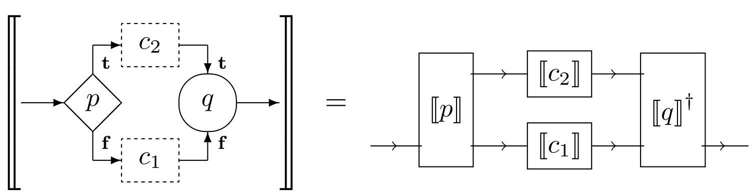


Intuitively, a decision separates an object into witnesses (in the first component) and counterexamples (in the second). As such, the partial inverse to a decision must be defined only on witnesses in the first component, and only on counterexamples in the second. But then, where decisions model *predicates*, codecisions (*i.e.*, partial inverses to decisions) model *assertions*.

With this in mind, we achieve a denotation of reversible conditionals as

$$[\![\textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q]\!] = [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![p]\!]$$

or, as diagrams



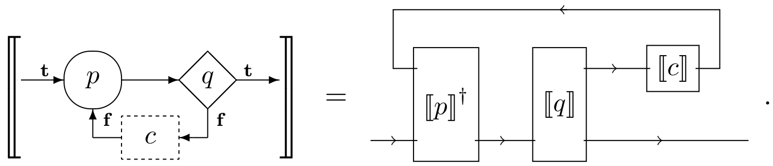
To give the denotation of reversible loops, we use the †-trace operator. Defining a shorthand for the body of the loop as

$$\beta[p, c, q] = (\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger$$

we obtain the denotation

$$[\![\textbf{from } p \textbf{ loop } c \textbf{ until } q]\!] = \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}(\beta[p, c, q]) = \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}((\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger)$$

or diagrammatically



That this has the desired operational behavior follows from the fact that the †-trace operator is canonically constructed in join inverse categories as

$$\mathrm{Tr}^U_{X,Y}(f) = f_{11} \vee \bigvee_{n \in \omega} f_{21} f_{22}^n f_{12} \ .$$

Recall that $f_{ij} = \amalg_j^\dagger f \amalg_i$. As such, for our loop construct defined above, the f_{11}-cases correpond to cases where a given state bypasses the loop entirely; $f_{21} f_{12}$ (that is, for $n = 0$) to cases where exactly one iteration is performed by a given state before exiting the loop; $f_{21} f_{22} f_{12}$ to cases where two iterations are performed before exiting; and so on. In this way, the given trace semantics contain all successive loop unrollings, as desired. We will make this more formal in the following section, where we show computational soundness and adequacy for these with respect to the operational semantics.

In order to be able to provide a correspondence between categorical and operational semantics, we also need an interpretation of the meta-command $\underline{loop}$. While it may not be so clear at the present, it turns out that the appropriate one is

$$[\![\underline{loop}[p, c, q]]\!] = \bigvee_{n \in \omega} \beta[p, c, q]_{21} \beta[p, c, q]_{22}^n \ .$$

While it may seem like a small point, the mere existence of a categorical semantics in inverse categories for a reversible programming language has some immediate benefits. In particular, that a programming language is reversible can be rather complicated to show by means of operational semantics (see, *e.g.*, [40, Sec. 2.3]), yet it follows directly in our categorical semantics, as it is compositional and all morphisms in inverse categories have a unique partial inverse.

6. Computational soundness and adequacy

Computational soundness and adequacy (see, *e.g.*, [13]) are the two fundamental properties of operational semantics with respect to their denotational counterparts, as soundness and completeness are for proof systems with respect to their semantics. In brief, computational soundness and adequacy state that the respective notions of *convergence* of the operational and denotational semantics are in agreement.

In the operational semantics, the notion of convergence seems straightforward: a program c converges in a state σ if there exists another state σ' such that $\sigma \vdash c \downarrow \sigma'$. On the denotational side, it seems less obvious what a proper notion of convergence is.

An idea (used by, *e.g.*, Fiore [13]) is to let values (in this case, states) be interpreted as *total* morphisms from some sufficiently simple object I into an appropriate object V (here, we will use our object Σ of states). In this context, the notion of convergence for a program p in a state σ is then that the resulting morphism $[\![p]\!]\,[\![\sigma]\!]$ is, again, (the denotation of) a state – *i.e.*, it is total. Naturally, this approach requires machinery to separate total maps from partial ones. As luck would have it inverse categories fit the bill perfectly, as they can be regarded as special instances of restriction categories.

To make this idea more clear in the current context, and to allow us to use the established formulations of computational soundness and adequacy, we define a model of a structured reversible flowchart language to be the following:

Definition 6.1. A model of a structured reversible flowchart language $\mathcal{L}$ consists of a join inverse category $\mathscr{C}$ with a disjointness tensor, further equipped with distinguished objects I and Σ satisfying

(i) the identity and zero maps on I are distinct, *i.e.*, $\mathrm{id}_I \neq 0_{I,I}$,
(ii) if $I \xrightarrow{e} I$ is a restriction idempotent then $e = \mathrm{id}_I$ or $e = 0_{I,I}$, and
(iii) each $\mathcal{L}$-state σ is interpreted as a *total* morphism $I \xrightarrow{[\![\sigma]\!]} \Sigma$.

Here, we think of I as the *indexing object*, and Σ as the *object of states*. In irreversible programming languages, the first two conditions in the definition above are often left out, as the indexing object is typically chosen to be the terminal object 1. However, terminal objects are degenerate in inverse categories, as they always coincide with the initial object when they exist – that is, they are zero objects. For this reason, we require instead the existence of a sufficiently simple indexing object, as described by these two properties. For example, in **PInj**, any one-element set will satisfy these conditions.

Even further, the third condition is typically proven rather than assumed. We include it here as an assumption since structured reversible flowchart languages may take many different forms, and we have no way of knowing how the concrete states are formed. As such, rather than limiting ourselves to languages where states take a certain form in order to show totality of interpretation, we instead assume it to be able to show properties about more programming languages.

This also leads us to another important point: We are only able to show computational soundness and adequacy for the operational semantics as they are stated, *i.e.*, we are not able to take into account the specific atomic steps (besides **skip**) or elementary predicates of the language.

As such, computational soundness and adequacy (and what may follow from that) should be understood *conditionally*: If a structured reversible flowchart language has a model of the form above *and* it is computationally sound and adequate with respect to its atomic steps and elementary predicates, then the entire interpretation is sound and adequate as well.

We begin by recalling the definition of the denotation of predicates and commands in a model of a structured reversible flowchart language from Section 5.

Definition 6.2. Recall the interpretation of predicates in $\mathcal{L}$ as decisions in $\mathscr{C}$:

(i) $[\![tt]\!] = \amalg_1$,
(ii) $[\![ff]\!] = \amalg_2$,
(iii) $[\![\textbf{not } p]\!] = \gamma\,[\![p]\!]$,
(iv) $[\![p \textbf{ and } q]\!] = \Big((\amalg_1 \overline{\amalg_1^\dagger\,[\![p]\!]}\;\overline{\amalg_1^\dagger\,[\![q]\!]}) \vee (\amalg_2(\overline{\amalg_2^\dagger\,[\![p]\!]} \vee \overline{\amalg_2^\dagger\,[\![q]\!]}))\Big)\,\overline{[\![p]\!]}\;\overline{[\![q]\!]}$.

Definition 6.3. Recall the interpretation of commands in $\mathcal{L}$ (and the meta-command $\underline{loop}$) as morphisms $\Sigma \to \Sigma$ in $\mathscr{C}$:

(i) $[\![\textbf{skip}]\!] = \mathrm{id}_\Sigma$,
(ii) $[\![c_1\ ;\ c_2]\!] = [\![c_2]\!]\,[\![c_1]\!]$,

(iii) $[\![\textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q]\!] = [\![q]\!]^\dagger ([\![c_1]\!] \oplus [\![c_2]\!])\, [\![p]\!]$,
(iv) $[\![\textbf{from } p \textbf{ loop } c \textbf{ until } q]\!] = \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}(\beta[p,c,q])$, and
(v) $\left[\!\left[\underline{loop}[p,c,q]\right]\!\right] = \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^n$

where $\beta[p,c,q] = (\mathrm{id}_\Sigma \oplus [\![c]\!])\, [\![q]\!]\, [\![p]\!]^\dagger : \Sigma\oplus\Sigma \to \Sigma\oplus\Sigma$. Note also that $f_{ij} = \amalg_j^\dagger f \amalg_i$.

The overall strategy we will use to show computational soundness and computational adequacy for programs is to start by showing it for predicates. To begin to tackle this, we first need a lemma regarding the totality of predicates.

Lemma 6.4. *Let p and q be $\mathcal{L}$-predicates. It is the case that*

(i) $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![tt]\!]} \Sigma\oplus\Sigma$ *and* $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![f\!f]\!]} \Sigma\oplus\Sigma$ *are total,*
(ii) $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![\textbf{not } p]\!]} \Sigma\oplus\Sigma$ *is total iff* $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![p]\!]} \Sigma\oplus\Sigma$ *is, and*
(iii) $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![p \textbf{ and } q]\!]} \Sigma\oplus\Sigma$ *is total iff* $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![p]\!]} \Sigma\oplus\Sigma$ *and* $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![q]\!]} \Sigma\oplus\Sigma$ *both are.*

Proof. For (i), it follows that

$$\overline{[\![tt]\!]\,[\![\sigma]\!]} = \overline{\amalg_1\,[\![\sigma]\!]} = \overline{\overline{\amalg_1}\,[\![\sigma]\!]} = \overline{\mathrm{id}_\Sigma\,[\![\sigma]\!]} = \overline{[\![\sigma]\!]} = \mathrm{id}_I,$$

where the final equality follows by the definition of a model. The case for $f\!f$ is entirely analogous.

For (ii), we have that

$$\overline{[\![\textbf{not } p]\!]\,[\![\sigma]\!]} = \overline{\gamma\,[\![p]\!]\,[\![\sigma]\!]} = \overline{\overline{\gamma}\,[\![p]\!]\,[\![\sigma]\!]} = \overline{\mathrm{id}_{\Sigma\oplus\Sigma}\,[\![p]\!]\,[\![\sigma]\!]} = \overline{[\![p]\!]\,[\![\sigma]\!]}$$

which implies directly that $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![\textbf{not } p]\!]} \Sigma\oplus\Sigma$ is total iff $I \xrightarrow{[\![\sigma]\!]} \Sigma \xrightarrow{[\![p]\!]} \Sigma\oplus\Sigma$ is.

For (iii), it suffices to show that $\overline{[\![p \textbf{ and } q]\!]\,[\![\sigma]\!]} = \overline{[\![p]\!]\,[\![\sigma]\!]}\;\overline{[\![q]\!]\,[\![\sigma]\!]}$, since $\overline{[\![p]\!]\,[\![\sigma]\!]} = \overline{[\![q]\!]\,[\![\sigma]\!]} = \mathrm{id}_I$ then yields $\overline{[\![p \textbf{ and } q]\!]\,[\![\sigma]\!]} = \mathrm{id}_I\mathrm{id}_I = \mathrm{id}_I$ directly; the other direction follows by the fact that if $\overline{g}\overline{f} = \mathrm{id}$ then $\mathrm{id} = \overline{g}\overline{f} = \overline{g}\overline{f}\;\overline{f} = \mathrm{id}\overline{f} = \overline{f}$ (and analogously for g).

We start by observing that

$$\overline{[\![p]\!]\,[\![\sigma]\!]} = \overline{((\amalg_1\overline{\amalg_1^\dagger\,[\![p]\!]}) \vee (\amalg_2\overline{\amalg_2^\dagger\,[\![p]\!]}))\,[\![\sigma]\!]} \tag{6.1}$$
$$= \overline{(\amalg_1\overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]}) \vee (\amalg_2\overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]})} \tag{6.2}$$
$$= \overline{(\amalg_1\,[\![\sigma]\!]\,\overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]}) \vee (\amalg_2\,[\![\sigma]\!]\,\overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]})} \tag{6.3}$$
$$= \overline{(\amalg_1\,[\![\sigma]\!]\,\overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]})} \vee \overline{(\amalg_2\,[\![\sigma]\!]\,\overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]})} \tag{6.4}$$
$$= \overline{(\overline{\amalg_1\,[\![\sigma]\!]}\overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]})} \vee \overline{(\overline{\amalg_2\,[\![\sigma]\!]}\overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]})} \tag{6.5}$$
$$= \overline{\overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]}} \vee \overline{\overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]}} \tag{6.6}$$
$$= \overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]} \vee \overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]} \tag{6.7}$$

where (6.1) follows by Lemma 4.7, (6.2) by distributivity of composition of joins, (6.3) by the fourth axiom of restriction categories (see Definition 3.1), (6.4) by distributivity of restriction over joins (see Definition 3.6), (6.5) by Lemma 3.3, (6.6) since $\overline{\amalg_1\,[\![\sigma]\!]} = \overline{\amalg_2\,[\![\sigma]\!]} = \mathrm{id}_I$ follows by (i), and (6.7) by Lemma 3.3. We may establish by analogous argument that

$$\overline{[\![q]\!]\,[\![\sigma]\!]} = \overline{\amalg_1^\dagger\,[\![q]\!]\,[\![\sigma]\!]} \vee \overline{\amalg_2^\dagger\,[\![q]\!]\,[\![\sigma]\!]}$$

as well. In the following, let $\sigma_p = \llbracket p \rrbracket \llbracket \sigma \rrbracket$ and $\sigma_q = \llbracket q \rrbracket \llbracket \sigma \rrbracket$. We start by computing

$$
\begin{aligned}
\llbracket p \textbf{ and } q \rrbracket \llbracket \sigma \rrbracket &= ((\amalg_1 \overline{\amalg_1^\dagger \llbracket p \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket}) \vee (\amalg_2 (\overline{\amalg_2^\dagger \llbracket p \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket}))) \overline{\llbracket p \rrbracket}\ \overline{\llbracket q \rrbracket} \llbracket \sigma \rrbracket && (6.8)\\
&= ((\amalg_1 \overline{\amalg_1^\dagger \llbracket p \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket}) \vee (\amalg_2 (\overline{\amalg_2^\dagger \llbracket p \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket}))) \llbracket \sigma \rrbracket\, \overline{\llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket} && (6.9)\\
&= ((\amalg_1 \overline{\amalg_1^\dagger \llbracket p \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket} \llbracket \sigma \rrbracket) \vee (\amalg_2 (\overline{\amalg_2^\dagger \llbracket p \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket}) \llbracket \sigma \rrbracket)) \overline{\sigma_p}\ \overline{\sigma_q} && (6.10)\\
&= ((\amalg_1 \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket \llbracket \sigma \rrbracket}) \vee \\
&\qquad (\amalg_2 \llbracket \sigma \rrbracket\, (\overline{\amalg_2^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket \llbracket \sigma \rrbracket}))) \overline{\sigma_p}\ \overline{\sigma_q} && (6.11)\\
&= ((\amalg_1 \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\amalg_2 \llbracket \sigma \rrbracket\, (\overline{\amalg_2^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_q}))) \overline{\sigma_p}\ \overline{\sigma_q} && (6.12)
\end{aligned}
$$

where (6.8) follows by the definition of $\llbracket p \textbf{ and } q \rrbracket$ (see Definition 6.2), (6.9) by two applications of the fourth axiom of restriction categories (see Definition 3.1), (6.10) by distributivity of composition over joins (see Definition 3.6) and the definitions of σ_p and σ_q, (6.11) by repeated applications of the fourth axiom of restriction categories, and (6.12) by definition of σ_p and σ_q. But then we have

$$
\begin{aligned}
\overline{\llbracket p \textbf{ and } q \rrbracket \llbracket \sigma \rrbracket} &= \overline{((\amalg_1 \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\amalg_2 \llbracket \sigma \rrbracket\, (\overline{\amalg_2^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_q}))) \overline{\sigma_p}\ \overline{\sigma_q}} && (6.13)\\
&= \overline{\overline{((\amalg_1 \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\amalg_2 \llbracket \sigma \rrbracket\, (\overline{\amalg_2^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_q})))} \overline{\sigma_p}\ \overline{\sigma_q}} && (6.14)\\
&= \overline{(\overline{(\amalg_1 \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q})} \vee \overline{(\amalg_2 \llbracket \sigma \rrbracket\, (\overline{\amalg_2^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_q}))}) \overline{\sigma_p}\ \overline{\sigma_q}} && (6.15)\\
&= \overline{((\overline{\amalg_1 \llbracket \sigma \rrbracket}\, \overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2 \llbracket \sigma \rrbracket} (\overline{\amalg_2^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_q}))) \overline{\sigma_p}\ \overline{\sigma_q}} && (6.16)\\
&= ((\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_q})) \overline{\sigma_p}\ \overline{\sigma_q} && (6.17)\\
&= (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}\ \overline{\sigma_p}\ \overline{\sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\sigma_p}\ \overline{\sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_q}\ \overline{\sigma_p}\ \overline{\sigma_q}) && (6.18)\\
&= (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}\ \overline{\sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\sigma_p}\ \overline{\sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_q}\ \overline{\sigma_q}\ \overline{\sigma_p}) && (6.19)\\
&= (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_q}\ \overline{\sigma_p}) && (6.20)\\
&= (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p} (\overline{\amalg_1^\dagger \sigma_q} \vee \overline{\amalg_2^\dagger \sigma_q})) \vee (\overline{\amalg_2^\dagger \sigma_q} (\overline{\amalg_1^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_p})) && (6.21)\\
&= (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\amalg_2^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_q}\ \overline{\amalg_1^\dagger \sigma_p}) \vee \\
&\qquad (\overline{\amalg_2^\dagger \sigma_q}\ \overline{\amalg_2^\dagger \sigma_p}) && (6.22)\\
&= (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_1^\dagger \sigma_p}\ \overline{\amalg_2^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\amalg_1^\dagger \sigma_q}) \vee (\overline{\amalg_2^\dagger \sigma_p}\ \overline{\amalg_2^\dagger \sigma_q}) && (6.23)\\
&= (\overline{\amalg_1^\dagger \sigma_p} \vee \overline{\amalg_2^\dagger \sigma_p})(\overline{\amalg_1^\dagger \sigma_q} \vee \overline{\amalg_2^\dagger \sigma_q}) = \overline{\sigma_p}\ \overline{\sigma_q} && (6.24)
\end{aligned}
$$

Here, (6.13) follows by (6.8)–(6.12), (6.14) by Lemma 3.3, (6.15) by distributivity of restriction over joins, (6.16) by the third axiom of restriction categories, (6.17) follows by (i), (6.18) by distributivity of composition over joins, (6.19) by commutativity of restriction idempotents (see Definition 3.1), (6.20) by Lemma 3.3, (6.21) by Lemma 4.7, (6.22) by distributivity of composition over joins, (6.23) by idempotence of joins, and (6.24) by distributivity of composition over joins and definition of σ_p and σ_q. $\square$

A common way to show computational soundness (see, *e.g.*, [13]) is to show a kind of preservation property; that interpretations are, in a sense, preserved across evaluation in the operational semantics. This is shown for predicates in the following lemma:

Lemma 6.5. *If* $\sigma \vdash p \rightsquigarrow b$ *then* $[\![p]\!]\,[\![\sigma]\!] = [\![b]\!]\,[\![\sigma]\!]$.

Proof. By induction on the structure of the derivation $\mathcal{D}$ of $\sigma \vdash p \rightsquigarrow b$.

- Case $\mathcal{D} = \dfrac{}{\sigma \vdash tt \rightsquigarrow tt}$. We trivially have $[\![tt]\!]\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!]$.
- Case $\mathcal{D} = \dfrac{}{\sigma \vdash \mathit{ff} \rightsquigarrow \mathit{ff}}$. Again, we trivially have $[\![\mathit{ff}]\!]\,[\![\sigma]\!] = [\![\mathit{ff}]\!]\,[\![\sigma]\!]$.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow tt}{\sigma \vdash \textbf{not}\ p \rightsquigarrow \mathit{ff}}$.

  By induction we have that $[\![p]\!]\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$. But then $[\![\textbf{not}\ p]\!]\,[\![\sigma]\!] = \gamma\,[\![p]\!]\,[\![\sigma]\!] = \gamma\,\amalg_1\,[\![\sigma]\!] = \amalg_2\,[\![\sigma]\!] = [\![\mathit{ff}]\!]\,[\![\sigma]\!]$.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow \mathit{ff}}{\sigma \vdash \textbf{not}\ p \rightsquigarrow tt}$.

  By induction we have that $[\![p]\!]\,[\![\sigma]\!] = [\![\mathit{ff}]\!]\,[\![\sigma]\!] = \amalg_2\,[\![\sigma]\!]$. Thus $[\![\textbf{not}\ p]\!]\,[\![\sigma]\!] = \gamma\,[\![p]\!]\,[\![\sigma]\!] = \gamma\,\amalg_2\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!]$.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash q \rightsquigarrow tt}{\sigma \vdash p\ \textbf{and}\ q \rightsquigarrow tt}$.

  By induction $[\![p]\!]\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$ and $[\![q]\!]\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$. We compute

$$
\begin{aligned}
[\![p\ \textbf{and}\ q]\!]\,[\![\sigma]\!] &= ((\amalg_1 \overline{\amalg_1^\dagger\,[\![p]\!]}\ \overline{\amalg_1^\dagger\,[\![q]\!]}) \vee (\amalg_2 (\overline{\amalg_2^\dagger\,[\![p]\!]} \vee \overline{\amalg_2^\dagger\,[\![q]\!]})))\overline{[\![p]\!]}\ \overline{[\![q]\!]}\,[\![\sigma]\!] && (6.25)\\
&= ((\amalg_1\,[\![\sigma]\!]\,\overline{\amalg_1^\dagger\,[\![p]\!]\,[\![\sigma]\!]}\ \overline{\amalg_1^\dagger\,[\![q]\!]\,[\![\sigma]\!]}) \vee \\
&\qquad (\amalg_2\,[\![\sigma]\!]\,(\overline{\amalg_2^\dagger\,[\![p]\!]\,[\![\sigma]\!]} \vee \overline{\amalg_2^\dagger\,[\![q]\!]\,[\![\sigma]\!]})))\overline{[\![p]\!]\,[\![\sigma]\!]}\ \overline{[\![q]\!]\,[\![\sigma]\!]} && (6.26)\\
&= ((\amalg_1\,[\![\sigma]\!]\,\overline{\amalg_1^\dagger\,\amalg_1\,[\![\sigma]\!]}\ \overline{\amalg_1^\dagger\,\amalg_1\,[\![\sigma]\!]}) \vee \\
&\qquad (\amalg_2\,[\![\sigma]\!]\,(\overline{\amalg_2^\dagger\,\amalg_1\,[\![\sigma]\!]} \vee \overline{\amalg_2^\dagger\,\amalg_1\,[\![\sigma]\!]})))\overline{\amalg_1\,[\![\sigma]\!]}\ \overline{\amalg_1\,[\![\sigma]\!]} && (6.27)\\
&= ((\amalg_1\,[\![\sigma]\!]\,\overline{[\![\sigma]\!]}\ \overline{[\![\sigma]\!]}) \vee (\amalg_2\,[\![\sigma]\!]\,(\overline{0_{\Sigma,\Sigma}\,[\![\sigma]\!]} \vee \overline{0_{\Sigma,\Sigma}\,[\![\sigma]\!]})))\overline{[\![\sigma]\!]}\ \overline{[\![\sigma]\!]} && (6.28)\\
&= ((\amalg_1\,[\![\sigma]\!]\,\overline{[\![\sigma]\!]}) \vee (\amalg_2\,[\![\sigma]\!]\,\overline{0_{\Sigma,\Sigma}\,[\![\sigma]\!]}))\overline{[\![\sigma]\!]} && (6.29)\\
&= ((\amalg_1\,[\![\sigma]\!]) \vee (\amalg_2\,[\![\sigma]\!]\,\overline{0_{I,\Sigma}}))\overline{[\![\sigma]\!]} && (6.30)\\
&= ((\amalg_1\,[\![\sigma]\!]) \vee (\amalg_2\,[\![\sigma]\!]\,0_{I,I}))\overline{[\![\sigma]\!]} && (6.31)\\
&= ((\amalg_1\,[\![\sigma]\!]) \vee 0_{I,\Sigma\oplus\Sigma})\overline{[\![\sigma]\!]} && (6.32)\\
&= \amalg_1\,[\![\sigma]\!]\,\overline{[\![\sigma]\!]} = \amalg_1\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!]\,. && (6.33)
\end{aligned}
$$

  where (6.25) is the definition of $[\![p\ \textbf{and}\ q]\!]$, (6.26) follows by (6.8) – (6.11), (6.27) by $[\![p]\!]\,[\![\sigma]\!] = [\![q]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$, (6.28) by Lemma 4.6, (6.29) by Lemma 3.3 and idempotence of joins, (6.30) by the first axiom of restriction categories and the universal mapping property of the zero object, (6.31) by the zero object a restriction zero, (6.32) once again by the universal mapping property of the zero object, and (6.33) by the first axiom of restriction categories and the definition of $[\![tt]\!]$.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow \mathit{ff} \qquad \sigma \vdash q \rightsquigarrow tt}{\sigma \vdash p\ \textbf{and}\ q \rightsquigarrow \mathit{ff}}$.

By induction $\llbracket p \rrbracket \llbracket \sigma \rrbracket = \llbracket ff \rrbracket \llbracket \sigma \rrbracket = \amalg_2 \llbracket \sigma \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma \rrbracket = \llbracket tt \rrbracket \llbracket \sigma \rrbracket = \amalg_1 \llbracket \sigma \rrbracket$.

$$\begin{aligned}
\llbracket p \textbf{ and } q \rrbracket \llbracket \sigma \rrbracket &= ((\amalg_1 \overline{\amalg_1^\dagger \llbracket p \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket}) \vee (\amalg_2 (\overline{\amalg_2^\dagger \llbracket p \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket}))) \overline{\llbracket p \rrbracket}\ \overline{\llbracket q \rrbracket} \llbracket \sigma \rrbracket \qquad (6.34) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket \llbracket \sigma \rrbracket}) \vee \\
&\quad (\amalg_2 \llbracket \sigma \rrbracket (\overline{\amalg_2^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket \llbracket \sigma \rrbracket}))) \overline{\llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket} \qquad (6.35) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket \overline{\amalg_1^\dagger \amalg_2 \llbracket \sigma \rrbracket}\ \overline{\amalg_1^\dagger \amalg_1 \llbracket \sigma \rrbracket}) \vee \\
&\quad (\amalg_2 \llbracket \sigma \rrbracket (\overline{\amalg_2^\dagger \amalg_2 \llbracket \sigma \rrbracket} \vee \overline{\amalg_2^\dagger \amalg_1 \llbracket \sigma \rrbracket}))) \overline{\amalg_2 \llbracket \sigma \rrbracket}\ \overline{\amalg_1 \llbracket \sigma \rrbracket} \qquad (6.36) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket \overline{0_{\Sigma,\Sigma} \llbracket \sigma \rrbracket}\ \overline{\llbracket \sigma \rrbracket}) \vee (\amalg_2 \llbracket \sigma \rrbracket (\overline{\llbracket \sigma \rrbracket} \vee \overline{0_{\Sigma,\Sigma} \llbracket \sigma \rrbracket}))) \overline{\llbracket \sigma \rrbracket}\ \overline{\llbracket \sigma \rrbracket} \qquad (6.37) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket 0_{I,I}\ \overline{\llbracket \sigma \rrbracket}) \vee (\amalg_2 \llbracket \sigma \rrbracket (\overline{\llbracket \sigma \rrbracket} \vee 0_{I,I}))) \overline{\llbracket \sigma \rrbracket} \qquad (6.38) \\
&= (0_{I,\Sigma \oplus \Sigma} \vee (\amalg_2 \llbracket \sigma \rrbracket \overline{\llbracket \sigma \rrbracket})) \overline{\llbracket \sigma \rrbracket} \qquad (6.39) \\
&= \amalg_2 \llbracket \sigma \rrbracket \overline{\llbracket \sigma \rrbracket}\ \overline{\llbracket \sigma \rrbracket} = \amalg_2 \llbracket \sigma \rrbracket = \llbracket ff \rrbracket \llbracket \sigma \rrbracket \qquad (6.40)
\end{aligned}$$

where (6.34) is the definition of $\llbracket p \textbf{ and } q \rrbracket$, (6.35) follows by (6.8) – (6.11), (6.36) by $\llbracket p \rrbracket \llbracket \sigma \rrbracket = \amalg_2 \llbracket \sigma \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma \rrbracket = \amalg_1 \llbracket \sigma \rrbracket$, (6.37) by Lemma 4.6, (6.38) by the first axiom of restriction categories as well as the universal mapping property of the zero object and the fact that it is a restriction zero, (6.39) by the universal mapping property of the zero object and the fact that zero maps are unit for joins, (6.40) by zero maps units for joins, the first axiom of restriction categories, and the definition of $\llbracket ff \rrbracket$.

- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash q \rightsquigarrow ff}{\sigma \vdash p \textbf{ and } q \rightsquigarrow ff}$, similar to the previous case.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow ff \qquad \sigma \vdash q \rightsquigarrow ff}{\sigma \vdash p \textbf{ and } q \rightsquigarrow ff}$.

By induction $\llbracket p \rrbracket \llbracket \sigma \rrbracket = \llbracket ff \rrbracket \llbracket \sigma \rrbracket = \amalg_2 \llbracket \sigma \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma \rrbracket = \llbracket ff \rrbracket \llbracket \sigma \rrbracket = \amalg_2 \llbracket \sigma \rrbracket$.

$$\begin{aligned}
\llbracket p \textbf{ and } q \rrbracket \llbracket \sigma \rrbracket &= ((\amalg_1 \overline{\amalg_1^\dagger \llbracket p \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket}) \vee (\amalg_2 (\overline{\amalg_2^\dagger \llbracket p \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket}))) \overline{\llbracket p \rrbracket}\ \overline{\llbracket q \rrbracket} \llbracket \sigma \rrbracket \qquad (6.41) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\amalg_1^\dagger \llbracket q \rrbracket \llbracket \sigma \rrbracket}) \vee \\
&\quad (\amalg_2 \llbracket \sigma \rrbracket (\overline{\amalg_2^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket} \vee \overline{\amalg_2^\dagger \llbracket q \rrbracket \llbracket \sigma \rrbracket}))) \overline{\llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket} \qquad (6.42) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket \overline{\amalg_1^\dagger \amalg_2 \llbracket \sigma \rrbracket}\ \overline{\amalg_1^\dagger \amalg_2 \llbracket \sigma \rrbracket}) \vee \\
&\quad (\amalg_2 \llbracket \sigma \rrbracket (\overline{\amalg_2^\dagger \amalg_2 \llbracket \sigma \rrbracket} \vee \overline{\amalg_2^\dagger \amalg_2 \llbracket \sigma \rrbracket}))) \overline{\amalg_2 \llbracket \sigma \rrbracket}\ \overline{\amalg_2 \llbracket \sigma \rrbracket} \qquad (6.43) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket \overline{0_{\Sigma,\Sigma} \llbracket \sigma \rrbracket}\ \overline{0_{\Sigma,\Sigma} \llbracket \sigma \rrbracket}) \vee (\amalg_2 \llbracket \sigma \rrbracket (\overline{\llbracket \sigma \rrbracket} \vee \overline{\llbracket \sigma \rrbracket}))) \overline{\llbracket \sigma \rrbracket}\ \overline{\llbracket \sigma \rrbracket} \qquad (6.44) \\
&= ((\amalg_1 \llbracket \sigma \rrbracket 0_{I,I}) \vee (\amalg_2 \llbracket \sigma \rrbracket \overline{\llbracket \sigma \rrbracket})) \overline{\llbracket \sigma \rrbracket} \qquad (6.45) \\
&= (0_{I,\Sigma \oplus \Sigma} \vee (\amalg_2 \llbracket \sigma \rrbracket \overline{\llbracket \sigma \rrbracket})) \overline{\llbracket \sigma \rrbracket} \qquad (6.46) \\
&= \amalg_2 \llbracket \sigma \rrbracket \overline{\llbracket \sigma \rrbracket}\ \overline{\llbracket \sigma \rrbracket} = \amalg_2 \llbracket \sigma \rrbracket = \llbracket ff \rrbracket \llbracket \sigma \rrbracket \qquad (6.47)
\end{aligned}$$

where (6.41) is the definition of $\llbracket p \textbf{ and } q \rrbracket \llbracket \sigma \rrbracket$, (6.42) follows by (6.8) – (6.11), (6.43) by $\llbracket p \rrbracket \llbracket \sigma \rrbracket = \llbracket q \rrbracket \llbracket \sigma \rrbracket = \amalg_2 \llbracket \sigma \rrbracket$, (6.44) by Lemma 4.6, (6.45) by idempotence of restriction idempotents and joins, (6.46) by the universal mapping property of the zero object, and (6.47) by the first axiom of restriction categories and the definition of $\llbracket ff \rrbracket$. □

With this done, the computational soundness lemma for predicates follows readily.

Lemma 6.6. *If there exists b such that $\sigma \vdash p \rightsquigarrow b$ then $[\![p]\!]\,[\![\sigma]\!]$ is total.*

Proof. Suppose there exists b such that $\sigma \vdash p \rightsquigarrow b$ by some derivation. It follows by the operational semantics that b must be either tt or ff, and in either case it follows by Lemma 6.4 (i) that $[\![b]\!]\,[\![\sigma]\!]$ is total, *i.e.*, $\overline{[\![b]\!]\,[\![\sigma]\!]} = \mathrm{id}_I$. Applying the derivation of $\sigma \vdash p \rightsquigarrow b$ to Lemma 6.5 yields that $[\![p]\!]\,[\![\sigma]\!] = [\![b]\!]\,[\![\sigma]\!]$, so specifically $\overline{[\![p]\!]\,[\![\sigma]\!]} = \overline{[\![b]\!]\,[\![\sigma]\!]} = \mathrm{id}_I$, as desired. □

Adequacy for predicates can then be shown by induction on the structure of the predicate, and by letting Lemma 6.4 (regarding the totality of predicates) do much of the heavy lifting.

Lemma 6.7. *If $[\![p]\!]\,[\![\sigma]\!]$ is total then there exists b such that $\sigma \vdash p \rightsquigarrow b$.*

Proof. By induction on the structure of p.

- Case $p = tt$. Then $\sigma \vdash tt \rightsquigarrow tt$ by $\dfrac{}{\sigma \vdash tt \rightsquigarrow tt}$.
- Case $p = ff$. Then $\sigma \vdash ff \rightsquigarrow ff$ by $\dfrac{}{\sigma \vdash ff \rightsquigarrow ff}$.
- Case $p = \mathbf{not}\ p'$. Since $[\![\mathbf{not}\ p']\!]\,[\![\sigma]\!]$ is total, it follows by Lemma 6.4 that $[\![p']\!]\,[\![\sigma]\!]$ is total as well, so by induction, there exists b such that $\sigma \vdash p' \rightsquigarrow b$ by some derivation $\mathcal{D}$. We have two cases to consider: If $b = tt$, $\mathcal{D}$ is a derivation of $\sigma \vdash p' \rightsquigarrow tt$, and so we may derive $\sigma \vdash \mathbf{not}\ p' \rightsquigarrow ff$ by
$$\frac{\overset{\mathcal{D}}{\sigma \vdash p' \rightsquigarrow tt}}{\sigma \vdash \mathbf{not}\ p' \rightsquigarrow ff}$$
If on the other hand $b = ff$, $\mathcal{D}$ is a derivation of $\sigma \vdash p' \rightsquigarrow ff$, and we may use the other **not**-rule with $\mathcal{D}$ to derive
$$\frac{\overset{\mathcal{D}}{\sigma \vdash p' \rightsquigarrow ff}}{\sigma \vdash \mathbf{not}\ p' \rightsquigarrow tt}\ .$$
- Case $p = q\ \mathbf{and}\ r$. Since we have that $[\![q\ \mathbf{and}\ r]\!]\,[\![\sigma]\!]$ is total, by Lemma 6.4, so are $[\![q]\!]\,[\![\sigma]\!]$ and $[\![r]\!]\,[\![\sigma]\!]$. Thus, it follows by induction that there exist b_1 and b_2 such that $\sigma \vdash q \rightsquigarrow b_1$ respectively $\sigma \vdash r \rightsquigarrow b_2$ by derivations $\mathcal{D}_1$ respectively $\mathcal{D}_2$. This gives us four cases depending on what b_1 and b_2 are. Luckily, these four cases match precisely the four different rules we have for **and**: For example, if $b_1 = tt$ and $b_2 = ff$, we may derive $\sigma \vdash q\ \mathbf{and}\ r \rightsquigarrow ff$ by
$$\frac{\overset{\mathcal{D}_1}{\sigma \vdash q \rightsquigarrow tt} \qquad \overset{\mathcal{D}_2}{\sigma \vdash r \rightsquigarrow ff}}{\sigma \vdash q\ \mathbf{and}\ r \rightsquigarrow ff},$$
and so on. □

With computational soundness and adequacy done for the predicates, we turn our attention to commands. Before we can show computational soundness, we will need a technical lemma regarding the denotational behaviour of loop bodies in states σ when the relevant predicates are either true or false (see Definition 6.3 for the definition of the loop body $\beta[p, c, q]$).

Lemma 6.8. *Let σ be a state, and p and q be predicates. Then*

(1) *If $\sigma \vdash p \rightsquigarrow tt$ and $\sigma \vdash q \rightsquigarrow tt$ then $\beta[p, c, q]_{11}\,[\![\sigma]\!] = [\![\sigma]\!]$,*
(2) *If $\sigma \vdash p \rightsquigarrow ff$ and $\sigma \vdash q \rightsquigarrow tt$ then $\beta[p, c, q]_{21}\,[\![\sigma]\!] = [\![\sigma]\!]$,*
(3) *If $\sigma \vdash p \rightsquigarrow tt$ and $\sigma \vdash q \rightsquigarrow ff$ then $\beta[p, c, q]_{12}\,[\![\sigma]\!] = [\![c]\!]\,[\![\sigma]\!]$, and*

(4) *If* $\sigma \vdash p \rightsquigarrow \mathit{ff}$ *and* $\sigma \vdash q \rightsquigarrow \mathit{ff}$ *then* $\beta[p,c,q]_{22} \llbracket\sigma\rrbracket = \llbracket c\rrbracket \llbracket\sigma\rrbracket$.
Further, in each case, for all other choices of i *and* j, $\beta[p,c,q]_{ij} \llbracket\sigma\rrbracket = 0_{I,\Sigma}$.

Proof. For (1), suppose $\sigma \vdash p \rightsquigarrow tt$ and $\sigma \vdash q \rightsquigarrow tt$, so by Lemma 6.5, $\llbracket p\rrbracket \llbracket\sigma\rrbracket = \llbracket tt\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$ and $\llbracket q\rrbracket \llbracket\sigma\rrbracket = \llbracket tt\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$. We have

$$\begin{aligned}
\beta[p,c,q]_{11} \llbracket\sigma\rrbracket &= \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket p\rrbracket^\dagger \amalg_1 \llbracket\sigma\rrbracket = \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket p\rrbracket^\dagger \llbracket p\rrbracket \llbracket\sigma\rrbracket && (6.48)\\
&= \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \overline{\llbracket p\rrbracket} \llbracket\sigma\rrbracket = \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket\sigma\rrbracket \overline{\llbracket p\rrbracket \llbracket\sigma\rrbracket} && (6.49)\\
&= \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \amalg_1 \llbracket\sigma\rrbracket \overline{\amalg_1 \llbracket\sigma\rrbracket} = \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \amalg_1 \llbracket\sigma\rrbracket && (6.50)\\
&= \amalg_1^\dagger \amalg_1 \mathrm{id}_\Sigma \llbracket\sigma\rrbracket = \overline{\amalg_1} \llbracket\sigma\rrbracket = \mathrm{id}_\Sigma \llbracket\sigma\rrbracket = \llbracket\sigma\rrbracket && (6.51)
\end{aligned}$$

where (6.48) follows by definition of $\beta[p,c,q]_{11}$ and $\llbracket p\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$, (6.49) by $\llbracket p\rrbracket$ a partial isomorphism and the fourth axiom of restriction categories, (6.50) by $\llbracket p\rrbracket \llbracket\sigma\rrbracket = \llbracket q\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$ and the first axiom of restriction categories, and (6.51) by naturality and totality of $\amalg_1$.

The proof of (2) is analogous to that of (1).

For (3), suppose $\sigma \vdash p \rightsquigarrow tt$ and $\sigma \vdash q \rightsquigarrow \mathit{ff}$, so by Lemma 6.5, $\llbracket p\rrbracket \llbracket\sigma\rrbracket = \llbracket tt\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$ and $\llbracket q\rrbracket \llbracket\sigma\rrbracket = \llbracket \mathit{ff}\rrbracket \llbracket\sigma\rrbracket = \amalg_2 \llbracket\sigma\rrbracket$. We compute

$$\begin{aligned}
\beta[p,c,q]_{12} \llbracket\sigma\rrbracket &= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket p\rrbracket^\dagger \amalg_1 \llbracket\sigma\rrbracket = \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket p\rrbracket^\dagger \llbracket p\rrbracket \llbracket\sigma\rrbracket && (6.52)\\
&= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \overline{\llbracket p\rrbracket} \llbracket\sigma\rrbracket = \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket\sigma\rrbracket \overline{\llbracket p\rrbracket \llbracket\sigma\rrbracket} && (6.53)\\
&= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \amalg_2 \llbracket\sigma\rrbracket \overline{\amalg_1 \llbracket\sigma\rrbracket} = \amalg_2^\dagger \amalg_2 \llbracket c\rrbracket \llbracket\sigma\rrbracket \overline{\overline{\amalg_1} \llbracket\sigma\rrbracket} && (6.54)\\
&= \overline{\amalg_2} \llbracket c\rrbracket \llbracket\sigma\rrbracket \overline{\mathrm{id}_\Sigma \llbracket\sigma\rrbracket} = \mathrm{id}_\Sigma \llbracket c\rrbracket \llbracket\sigma\rrbracket \overline{\llbracket\sigma\rrbracket} = \llbracket c\rrbracket \llbracket\sigma\rrbracket && (6.55)
\end{aligned}$$

where (6.52) follows by definition of $\beta[p,c,q]_{12}$ and $\llbracket p\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$, (6.53) by $\llbracket p\rrbracket$ a partial isomorphism and the fourth axiom of restriction categories, (6.54) by $\llbracket q\rrbracket \llbracket\sigma\rrbracket = \amalg_2 \llbracket\sigma\rrbracket$, naturality of $\amalg_2$ and Lemma 3.3, and (6.55) by totality of $\amalg_2$ and the first axiom of restriction categories.

The proof of (4) is analogous.

To see that in each case, for all other choices of i, j, $\beta[p,c,q]_{ij} \llbracket\sigma\rrbracket = 0_{I,\Sigma}$, we show a few of the cases where $\sigma \vdash p \rightsquigarrow tt$ and $\sigma \vdash q \rightsquigarrow tt$. The rest follow by the same line of reasoning. Recall that when $\sigma \vdash p \rightsquigarrow tt$ and $\sigma \vdash q \rightsquigarrow tt$ we have $\llbracket p\rrbracket \llbracket\sigma\rrbracket = \llbracket tt\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$ and $\llbracket q\rrbracket \llbracket\sigma\rrbracket = \llbracket tt\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$.

$$\begin{aligned}
\beta[p,c,q]_{12} \llbracket\sigma\rrbracket &= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket p\rrbracket^\dagger \amalg_1 \llbracket\sigma\rrbracket = \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket p\rrbracket^\dagger \llbracket p\rrbracket \llbracket\sigma\rrbracket && (6.56)\\
&= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \overline{\llbracket p\rrbracket} \llbracket\sigma\rrbracket = \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \llbracket q\rrbracket \llbracket\sigma\rrbracket \overline{\llbracket p\rrbracket \llbracket\sigma\rrbracket} && (6.57)\\
&= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c\rrbracket) \amalg_1 \llbracket\sigma\rrbracket \overline{\llbracket p\rrbracket \llbracket\sigma\rrbracket} = \amalg_2^\dagger \amalg_1 \mathrm{id}_\Sigma \llbracket\sigma\rrbracket \overline{\llbracket p\rrbracket \llbracket\sigma\rrbracket} && (6.58)\\
&= 0_{\Sigma,\Sigma} \mathrm{id}_\Sigma \llbracket\sigma\rrbracket \overline{\llbracket p\rrbracket \llbracket\sigma\rrbracket} = 0_{I,\Sigma} && (6.59)
\end{aligned}$$

where (6.56) and (6.57) follow as in (6.52) and (6.53), (6.58) by $\llbracket q\rrbracket \llbracket\sigma\rrbracket = \amalg_1 \llbracket\sigma\rrbracket$ and naturality of $\amalg_1$, and (6.59) by Lemma 4.6 and the universal mapping property of the zero object.

Finally, for $\beta[p,c,q]_{21}$ we have

$$\beta[p,c,q]_{21}\,[\![\sigma]\!] = \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger\,\amalg_2\,[\![\sigma]\!] = \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger\,\gamma\,\amalg_1\,[\![\sigma]\!] \tag{6.60}$$

$$= \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger\,\gamma\,[\![p]\!]\,[\![\sigma]\!] = \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger\,[\![\mathbf{not}\ p]\!]\,[\![\sigma]\!] \tag{6.61}$$

$$= \amalg_1^\dagger(\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,0_{\Sigma,\Sigma}\,[\![\sigma]\!] = 0_{I,\Sigma} \tag{6.62}$$

where (6.60) follows by definition of $\beta[p,c,q]_{21}$ and $\gamma\amalg_1 = \amalg_2$, (6.61) by $[\![p]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$ and definition of $[\![\mathbf{not}\ p]\!]$, and (6.62) by $[\![p]\!]^\dagger\,[\![\mathbf{not}\ p]\!] = 0_{\Sigma,\Sigma}$ and the universal mapping property of the zero object. $\square$

With this lemma done, we turn our attention to the preservation lemma for commands in order to show computational soundness.

Lemma 6.9. *If $\sigma \vdash c \downarrow \sigma'$ then $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$.*

Proof. By induction on the structure of the derivation $\mathcal{D}$ of $\sigma \vdash c \downarrow \sigma'$.

- Case $\mathcal{D} = \dfrac{}{\sigma \vdash \mathbf{skip} \downarrow \sigma}$. We have $[\![\mathbf{skip}]\!]\,[\![\sigma]\!] = \mathrm{id}_\Sigma\,[\![\sigma]\!] = [\![\sigma]\!]$.
- Case $\mathcal{D} = \dfrac{\sigma \vdash c_1 \downarrow \sigma' \qquad \sigma' \vdash c_2 \downarrow \sigma''}{\sigma \vdash c_1\ ;\ c_2 \downarrow \sigma''}$.

  By induction, $[\![c_1]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$ and $[\![c_2]\!]\,[\![\sigma']\!] = [\![\sigma'']\!]$. But then
  $$[\![c_1\ ;\ c_2]\!]\,[\![\sigma]\!] = [\![c_2]\!]\,[\![c_1]\!]\,[\![\sigma]\!] = [\![c_2]\!]\,[\![\sigma']\!] = [\![\sigma'']\!]$$
  as desired.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow tt \qquad \sigma \vdash c_1 \downarrow \sigma' \qquad \sigma' \vdash q \rightsquigarrow tt}{\sigma \vdash \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q \downarrow \sigma'}$.

  By induction, $[\![c_1]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$, and by Lemma 6.5, $[\![p]\!]\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$ and $[\![q]\!]\,[\![\sigma']\!] = [\![tt]\!]\,[\![\sigma']\!] = \amalg_1\,[\![\sigma']\!]$. We compute:
  $$[\![\mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![p]\!]\,[\![\sigma]\!] \tag{6.63}$$
  $$= [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,\amalg_1\,[\![\sigma]\!] \tag{6.64}$$
  $$= [\![q]\!]^\dagger\,\amalg_1\,[\![c_1]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,\amalg_1\,[\![\sigma']\!] = [\![q]\!]^\dagger\,[\![q]\!]\,[\![\sigma']\!] \tag{6.65}$$
  $$= \overline{[\![q]\!]}\,[\![\sigma']\!] = [\![\sigma']\!]\,\overline{[\![q]\!]\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{\amalg_1\,[\![\sigma']\!]} \tag{6.66}$$
  $$= [\![\sigma']\!]\,\overline{\overline{\amalg_1}\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{\mathrm{id}_\Sigma\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{[\![\sigma']\!]} = [\![\sigma']\!] \tag{6.67}$$
  where (6.63) follows by definition of $[\![\mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q]\!]$; (6.64) by $[\![p]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$; (6.65) by naturality of $\amalg_1$, $[\![c_1]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$, and $[\![q]\!]\,[\![\sigma']\!] = \amalg_1\,[\![\sigma']\!]$; (6.66) by $[\![q]\!]$ a partial isomorphism, the fourth axiom of restriction categories, and $[\![q]\!]\,[\![\sigma']\!] = \amalg_1\,[\![\sigma']\!]$; and (6.67) by Lemmas 3.3 and 4.6 and the first axiom of restriction categories.
- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow f\!f \qquad \sigma \vdash c_2 \downarrow \sigma' \qquad \sigma' \vdash q \rightsquigarrow f\!f}{\sigma \vdash \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q \downarrow \sigma'}$.

  By induction, $[\![c_2]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$, and by Lemma 6.5, $[\![p]\!]\,[\![\sigma]\!] = [\![f\!f]\!]\,[\![\sigma]\!] = \amalg_2\,[\![\sigma]\!]$ and

$\llbracket q \rrbracket \llbracket \sigma' \rrbracket = \llbracket \mathit{ff} \rrbracket \llbracket \sigma' \rrbracket = \amalg_2 \llbracket \sigma' \rrbracket$. We have

$$\begin{aligned}
\llbracket \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q \rrbracket \llbracket \sigma \rrbracket &= \llbracket q \rrbracket^\dagger \left(\llbracket c_1 \rrbracket \oplus \llbracket c_2 \rrbracket \right) \llbracket p \rrbracket \llbracket \sigma \rrbracket && (6.68)\\
&= \llbracket q \rrbracket^\dagger \left(\llbracket c_1 \rrbracket \oplus \llbracket c_2 \rrbracket \right) \amalg_2 \llbracket \sigma \rrbracket && (6.69)\\
&= \llbracket q \rrbracket^\dagger \amalg_2 \llbracket c_2 \rrbracket \llbracket \sigma \rrbracket = \llbracket q \rrbracket^\dagger \amalg_2 \llbracket \sigma' \rrbracket = \llbracket q \rrbracket^\dagger \llbracket q \rrbracket \llbracket \sigma' \rrbracket && (6.70)\\
&= \overline{\llbracket q \rrbracket} \llbracket \sigma' \rrbracket = \llbracket \sigma' \rrbracket \overline{\llbracket q \rrbracket \llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket \overline{\amalg_2 \llbracket \sigma' \rrbracket} && (6.71)\\
&= \llbracket \sigma' \rrbracket \overline{\overline{\amalg_2} \llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket \overline{\mathrm{id}_\Sigma \llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket \overline{\llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket . && (6.72)
\end{aligned}$$

which follows by similar arguments as in (6.63) – (6.67).

- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow \mathit{tt} \qquad \sigma \vdash q \rightsquigarrow \mathit{tt}}{\sigma \vdash \mathbf{from}\ p\ \mathbf{loop}\ c\ \mathbf{until}\ q \downarrow \sigma}$.

  Since $\sigma \vdash p \rightsquigarrow \mathit{tt}$ and $\sigma \vdash q \rightsquigarrow \mathit{tt}$, by Lemma 6.8 we get $\beta[p,c,q]_{11} \llbracket \sigma \rrbracket = \llbracket \sigma \rrbracket$ and $\beta[p,c,q]_{12} \llbracket \sigma \rrbracket = 0_{I,\Sigma}$, and so

$$\begin{aligned}
\llbracket \mathbf{from}\ p\ \mathbf{loop}\ c\ \mathbf{until}\ q \rrbracket \llbracket \sigma \rrbracket &= \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}(\beta[p,c,q]) && (6.73)\\
&= \left(\beta[p,c,q]_{11} \vee \bigvee_{n \in \omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^n\ \beta[p,c,q]_{12} \right) \llbracket \sigma \rrbracket && (6.74)\\
&= \left((\beta[p,c,q]_{11} \llbracket \sigma \rrbracket) \vee \bigvee_{n \in \omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^n\ \beta[p,c,q]_{12} \llbracket \sigma \rrbracket \right) && (6.75)\\
&= \llbracket \sigma \rrbracket \vee \bigvee_{n \in \omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^n\ 0_{I,\Sigma} = \llbracket \sigma \rrbracket \vee 0_{I,\Sigma} = \llbracket \sigma \rrbracket && (6.76)
\end{aligned}$$

  where (6.73) follows by definition of $\llbracket \mathbf{from}\ p\ \mathbf{loop}\ c\ \mathbf{until}\ q \rrbracket$, (6.74) by definition of the canonical trace in join inverse categories (see Proposition 3.10), (6.75) by distributivity of composition over joins, and (6.76) by $\beta[p,c,q]_{11} \llbracket \sigma \rrbracket = \llbracket \sigma \rrbracket$, $\beta[p,c,q]_{12} \llbracket \sigma \rrbracket = 0_{I,\Sigma}$, the universal mapping property of the zero object, and the fact that zero maps are units for joins.

- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow \mathit{tt} \qquad \sigma \vdash q \rightsquigarrow \mathit{ff} \qquad \sigma \vdash c \downarrow \sigma' \qquad \sigma' \vdash \underline{\mathit{loop}}[p,c,q] \downarrow \sigma''}{\sigma \vdash \mathbf{from}\ p\ \mathbf{loop}\ c\ \mathbf{until}\ q \downarrow \sigma''}$.

  By induction, $\llbracket c \rrbracket \llbracket \sigma \rrbracket = \llbracket \sigma' \rrbracket$ and $\left\llbracket \underline{\mathit{loop}}[p,c,q] \right\rrbracket \llbracket \sigma' \rrbracket = \llbracket \sigma'' \rrbracket$, and since $\sigma \vdash p \rightsquigarrow \mathit{tt}$ and

$\sigma \vdash q \rightsquigarrow \mathit{ff}$, by Lemma 6.8 we get $\beta[p,c,q]_{11} [\![\sigma]\!] = 0_{I,\Sigma}$ and $\beta[p,c,q]_{12} [\![\sigma]\!] = [\![c]\!] [\![\sigma]\!]$. Thus

$$[\![\textbf{from } p \textbf{ loop } c \textbf{ until } q]\!] [\![\sigma]\!] = \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}(\beta[p,c,q]) [\![\sigma]\!] \tag{6.77}$$
$$= \left(\beta[p,c,q]_{11} \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^{n}\ \beta[p,c,q]_{12}\right) [\![\sigma]\!] \tag{6.78}$$
$$= \left((\beta[p,c,q]_{11} [\![\sigma]\!]) \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^{n}\ \beta[p,c,q]_{12} [\![\sigma]\!]\right) \tag{6.79}$$
$$= \left(0_{I,\Sigma} \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^{n}\ [\![c]\!] [\![\sigma]\!]\right) \tag{6.80}$$
$$= \bigvee_{n\in\omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^{n}\ [\![\sigma']\!] \tag{6.81}$$
$$= \left(\bigvee_{n\in\omega} \beta[p,c,q]_{21}\ \beta[p,c,q]_{22}^{n}\right) [\![\sigma']\!] \tag{6.82}$$
$$= [\![\underline{loop}[p,c,q]]\!]\ [\![\sigma']\!] = [\![\sigma'']\!] \tag{6.83}$$

where (6.77) follows by the definition of $[\![\textbf{from } p \textbf{ loop } c \textbf{ until } q]\!]$, (6.78) by the definition of the canonical trace (Proposition 3.10), (6.79) by distributivity of composition over joins, (6.80) by $\beta[p,c,q]_{11} [\![\sigma]\!] = 0_{I,\Sigma}$ and $\beta[p,c,q]_{12} [\![\sigma]\!] = [\![c]\!] [\![\sigma]\!]$, (6.81) by $[\![\sigma']\!] = [\![c]\!] [\![\sigma]\!]$ and the fact that zero maps are units for joins, (6.82) by distributivity of composition over joins, and (6.83) by definition of $[\![\underline{loop}[p,c,q]]\!]$ and $[\![\underline{loop}[p,c,q]]\!]\ [\![\sigma']\!] = [\![\sigma'']\!]$.

- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow \mathit{ff} \qquad \sigma \vdash q \rightsquigarrow \mathit{tt}}{\sigma \vdash \underline{loop}[p,c,q] \downarrow \sigma}$.

Since $\sigma \vdash p \rightsquigarrow \mathit{ff}$ and $\sigma \vdash q \rightsquigarrow \mathit{tt}$, by Lemma 6.8 we get $\beta[p,c,q]_{22} [\![\sigma]\!] = 0_{I,\Sigma}$ and $\beta[p,c,q]_{21} [\![\sigma]\!] = [\![\sigma]\!]$. This gives us

$$[\![\underline{loop}[p,c,q]]\!]\ [\![\sigma]\!] = \left(\bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\right) [\![\sigma]\!] \tag{6.84}$$
$$= \left(\beta[p,c,q]_{21} \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n+1}\right) [\![\sigma]\!] \tag{6.85}$$
$$= \left((\beta[p,c,q]_{21} [\![\sigma]\!]) \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n+1} [\![\sigma]\!]\right) \tag{6.86}$$
$$= \left([\![\sigma]\!] \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\beta[p,c,q]_{22} [\![\sigma]\!]\right) \tag{6.87}$$
$$= \left([\![\sigma]\!] \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}0_{I,\Sigma}\right) \tag{6.88}$$
$$= [\![\sigma]\!] \vee 0_{I,\Sigma} = [\![\sigma]\!] \tag{6.89}$$

where (6.84) follows by definition of $[\![\underline{loop}[p,c,q]]\!]$, (6.85) by unrolling the case for $n = 0$, (6.86) by distributivity of composition over joins, (6.87) by $\beta[p,c,q]_{21} [\![\sigma]\!] = [\![\sigma]\!]$ and unrolling of the $n+1$-ary composition, (6.88) by $\beta[p,c,q]_{22} [\![\sigma]\!] = 0_{I,\Sigma}$, and (6.89) by the universal mapping property of the zero object and the fact that zero maps are units for joins.

- Case $\mathcal{D} = \dfrac{\sigma \vdash p \rightsquigarrow \mathit{ff} \quad \sigma \vdash q \rightsquigarrow \mathit{ff} \quad \sigma \vdash c \downarrow \sigma' \quad \sigma' \vdash \underline{loop}[p,c,q] \downarrow \sigma''}{\sigma \vdash \underline{loop}[p,c,q] \downarrow \sigma''}$.

  By induction, $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$ and $[\![\underline{loop}[p,c,q]]\!]\,[\![\sigma']\!] = [\![\sigma'']\!]$, and since $\sigma \vdash p \rightsquigarrow \mathit{ff}$ and $\sigma \vdash q \rightsquigarrow \mathit{ff}$, it follows by Lemma 6.8 that $\beta[p,c,q]_{22}\,[\![\sigma]\!] = [\![c]\!]\,[\![\sigma]\!]$ and $\beta[p,c,q]_{21}\,[\![\sigma]\!] = 0_{I,\Sigma}$. We then have

$$
\begin{aligned}
[\![\underline{loop}[p,c,q]]\!]\,[\![\sigma]\!] &= \left(\bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\right)[\![\sigma]\!] && (6.90)\\
&= \left(\beta[p,c,q]_{21} \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n+1}\right)[\![\sigma]\!] && (6.91)\\
&= \left((\beta[p,c,q]_{21}\,[\![\sigma]\!]) \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n+1}\,[\![\sigma]\!]\right) && (6.92)\\
&= 0_{I,\Sigma} \vee \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n+1}\,[\![\sigma]\!] && (6.93)\\
&= \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\beta[p,c,q]_{22}\,[\![\sigma]\!] && (6.94)\\
&= \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\,[\![c]\!]\,[\![\sigma]\!] && (6.95)\\
&= \bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\,[\![\sigma']\!] && (6.96)\\
&= \left(\bigvee_{n\in\omega} \beta[p,c,q]_{21}\beta[p,c,q]_{22}^{n}\right)[\![\sigma']\!] && (6.97)\\
&= [\![\underline{loop}[p,c,q]]\!]\,[\![\sigma']\!] = [\![\sigma'']\!] && (6.98)
\end{aligned}
$$

  where (6.90) follows by definition of $[\![\underline{loop}[p,c,q]]\!]$, (6.91) by unrolling the case for $n = 0$, (6.92) by distributivity of composition over joins, (6.93) by $\beta[p,c,q]_{21}\,[\![\sigma]\!] = 0_{I,\Sigma}$, (6.94) by zero maps units for joins and unrolling of the $n+1$-ary composition, (6.95) by $\beta[p,c,q]_{22}\,[\![\sigma]\!] = [\![c]\!]\,[\![\sigma]\!]$, (6.96) by $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$, (6.97) by distributivity of composition over joins, and finally (6.98) by definition of $[\![\underline{loop}[p,c,q]]\!]$ and $[\![\underline{loop}[p,c,q]]\!]\,[\![\sigma']\!] = [\![\sigma'']\!]$. □

This finally allows us to show the computational soundness theorem for commands – and so, for programs – in a straightforward manner.

Theorem 6.10 (Computational soundness)**.** *If there exists σ' such that $\sigma \vdash c \downarrow \sigma'$ then $[\![c]\!]\,[\![\sigma]\!]$ is total.*

Proof. Suppose there exists σ' such that $\sigma \vdash c \downarrow \sigma'$. By Lemma 6.9, $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$, and since the interpretation of *any* state is assumed to be total, it follows that $\overline{[\![c]\!]\,[\![\sigma]\!]} = \overline{[\![\sigma']\!]} = \mathrm{id}_I$, which was what we wanted. □

With computational soundness done, we only have computational adequacy left to prove. Adequacy is much simpler than usual in our case, as we have no higher order data to deal with, and as such, it can be shown by plain structural induction rather than by the assistance of logical relations. Nevertheless, we require two technical lemmas in order to succeed.

Lemma 6.11. *Let $f : A \oplus U \to B \oplus U$ and $s : I \to A$ be morphisms. If $\mathrm{Tr}^U_{A,B}(f)s$ is total, either $\mathrm{Tr}^U_{A,B}(f)s = f_{11}s$, or there exists $n \in \omega$ such that $\mathrm{Tr}^U_{A,B}(f)s = f_{21}f_{22}^n f_{12}s$.*

Proof. Since the trace is canonically constructed, it takes the form

$$\mathrm{Tr}^U_{A,B}(f) = f_{11} \vee \bigvee_{n\in\omega} f_{21}f_{22}^n f_{12}.$$

Further, in the proof of Theorem 20 in [26], it is shown that this join not only exists but is a *disjoint* join, *i.e.*, for any choice of $n \in \omega$,

$$f_{11}\overline{f_{21}f_{22}^n f_{12}} = (f_{21}f_{22}^n f_{12})\,\overline{f_{11}} = 0_{A,B}$$

and for all $n, m \in \omega$ with $n \neq m$,

$$\overline{f_{21}f_{22}^n f_{12}}f_{21}f_{22}^m f_{12} = \overline{f_{21}f_{22}^m f_{12}}f_{21}f_{22}^n f_{12} = 0_{A,B}.$$

But then,

$$\begin{aligned}\overline{\mathrm{Tr}^U_{A,B}(f)s} &= \overline{\left(f_{11} \vee \bigvee_{n\in\omega} f_{21}f_{22}^n f_{12}\right)s} \\ &= \overline{(f_{11}s) \vee \bigvee_{n\in\omega} (f_{21}f_{22}^n f_{12}s)} \\ &= \overline{f_{11}s} \vee \bigvee_{n\in\omega} \overline{f_{21}f_{22}^n f_{12}s}.\end{aligned}$$

Since all of the morphisms $\overline{f_{11}s}$ and $\overline{f_{21}f_{22}^n f_{12}s}$ for any $n \in \omega$ are restriction idempotents $I \to I$, it follows for each of them that they are either equal to id_I or to $0_{I,I}$. Suppose that none of these are equal to the identity id_I. Then they must all be $0_{I,I}$, and so $\overline{\mathrm{Tr}^U_{A,B}(f)s} = 0_{I,I} \neq \mathrm{id}_I$, contradicting totality. On the other hand, suppose that there exists an identity among these. Then, it follows by the disjointness property above that the rest must be $0_{I,I}$. □

With these done, we are finally ready to tackle the adequacy theorem.

Theorem 6.12 (Computational adequacy)**.** *If $[\![c]\!]\,[\![\sigma]\!]$ is total then there exists σ' such that $\sigma \vdash c \downarrow \sigma'$.*

Proof. By induction on the structure of c.

- Case $c = \mathbf{skip}$. Then $\sigma \vdash \mathbf{skip} \downarrow \sigma$ by $\dfrac{}{\sigma \vdash \mathbf{skip} \downarrow \sigma}$.
- Case $c = c_1 \ ; \ c_2$.

  In this case, $[\![c]\!]\,[\![\sigma]\!] = [\![c_1 \ ; \ c_2]\!]\,[\![\sigma]\!] = [\![c_2]\!]\,[\![c_1]\!]\,[\![\sigma]\!]$. Since this is total, so is $[\![c_1]\!]\,[\![\sigma]\!]$ by Lemma 3.3. But then, by induction, there exists σ' such that $\sigma \vdash c_1 \downarrow \sigma'$ by some derivation $\mathcal{D}_1$, and by Lemma 6.9, $[\![c_1]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$. But then $[\![c_2]\!]\,[\![c_1]\!]\,[\![\sigma]\!] = [\![c_2]\!]\,[\![\sigma']\!]$, so by induction there exists σ'' such that $\sigma' \vdash c_2 \downarrow \sigma''$ by some derivation $\mathcal{D}_2$. But then $\sigma \vdash c_1 \ ; \ c_2 \downarrow \sigma''$ by

$$\frac{\overset{\mathcal{D}_1}{\sigma \vdash c_1 \downarrow \sigma'} \qquad \overset{\mathcal{D}_2}{\sigma' \vdash c_2 \downarrow \sigma''}}{\sigma \vdash c_1 \ ; \ c_2 \downarrow \sigma''}\ .$$

- Case $c = \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q$.

  Thus, $[\![c]\!]\,[\![\sigma]\!] = [\![\mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![p]\!]\,[\![\sigma]\!]$, and since this is total, $[\![p]\!]\,[\![\sigma]\!]$ is total as well by analogous argument to the previous case. It then follows by Lemma 6.7 that there exists b such that $\sigma \vdash p \rightsquigarrow b$ by some derivation $\mathcal{D}_1$, and by Lemma 6.5, $[\![p]\!]\,[\![\sigma]\!] = [\![b]\!]\,[\![\sigma]\!]$. We have two cases depending on what b is.

  When $b = tt$ we have

$$\begin{aligned}[\![c]\!]\,[\![\sigma]\!] &= [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![p]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![tt]\!]\,[\![\sigma]\!] \\ &= [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,\amalg_1\,[\![\sigma]\!] = [\![q]\!]^\dagger\,\amalg_1\,[\![c_1]\!]\,[\![\sigma]\!]\end{aligned}$$

  by $[\![p]\!]\,[\![\sigma]\!] = [\![tt]\!]\,[\![\sigma]\!] = \amalg_1\,[\![\sigma]\!]$ and naturality of $\amalg_1$. Since this is total, $[\![c_1]\!]\,[\![\sigma]\!]$ must be total as well by Lemma 3.3. But then, by induction, there exists σ' such that $\sigma \vdash c_1 \downarrow \sigma'$ by some derivation $\mathcal{D}_2$, and by Lemma 6.9, $[\![c_1]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$. Continuing the computation, we get

$$\begin{aligned}[\![c]\!]\,[\![\sigma]\!] &= [\![q]\!]^\dagger\,\amalg_1\,[\![c_1]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,\amalg_1\,[\![\sigma']\!] = (\amalg_1^\dagger\,[\![q]\!])^\dagger\,[\![\sigma']\!] = \overline{\amalg_1^\dagger\,[\![q]\!]}^\dagger\,[\![\sigma']\!] \\ &= \overline{\amalg_1^\dagger\,[\![q]\!]}\,[\![\sigma']\!] = [\![\sigma']\!]\,\overline{\amalg_1^\dagger\,[\![q]\!]\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{\amalg_1^\dagger\,[\![q]\!]\,[\![\sigma']\!]}\,\overline{[\![q]\!]\,[\![\sigma']\!]}\end{aligned}$$

  where we exploit that $[\![c_1]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$ as well as the fact that $\amalg_1^\dagger\,[\![q]\!] = \overline{\amalg_1^\dagger\,[\![q]\!]}$ (by Lemma 4.7) and the fact that restriction idempotents are their own partial inverses. But then $\overline{[\![q]\!]\,[\![\sigma']\!]}$ must be total, in turn meaning that $[\![q]\!]\,[\![\sigma']\!]$ must be total. But then by Lemma 6.7, there must exist b' such that $\sigma' \vdash q \rightsquigarrow b'$ by some derivation $\mathcal{D}_3$, with $[\![q]\!]\,[\![\sigma']\!] = [\![b']\!]\,[\![\sigma']\!]$ by Lemma 6.5. Again, we have two cases depending on b'. If $b' = tt$, we derive $\sigma \vdash \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q \downarrow \sigma'$ by

$$\frac{\overset{\mathcal{D}_1}{\sigma \vdash p \rightsquigarrow tt} \qquad \overset{\mathcal{D}_2}{\sigma \vdash c_1 \downarrow \sigma'} \qquad \overset{\mathcal{D}_3}{\sigma' \vdash q \rightsquigarrow tt}}{\sigma \vdash \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q \downarrow \sigma'}$$

  On the other hand, when $b' = f\!f$, we have

$$\begin{aligned}[\![c]\!]\,[\![\sigma]\!] &= [\![\sigma']\!]\,\overline{\amalg_1^\dagger\,[\![q]\!]\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{\amalg_1^\dagger\,[\![f\!f]\!]\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{\amalg_1^\dagger\,\amalg_2\,[\![\sigma']\!]} \\ &= [\![\sigma']\!]\,\overline{0_{\Sigma,\Sigma}\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{0_{I,\Sigma}} = [\![\sigma']\!]\,0_{I,I} = 0_{I,\Sigma}\end{aligned}$$

  using $[\![q]\!]\,[\![\sigma']\!] = [\![f\!f]\!]\,[\![\sigma']\!] = \amalg_2\,[\![\sigma']\!]$ and $\amalg_1^\dagger\amalg_2 = 0_{\Sigma,\Sigma}$ by Lemma 4.6. But then $\overline{[\![c]\!]\,[\![\sigma]\!]} = 0_{I,I}$, contradicting $\overline{[\![c]\!]\,[\![\sigma]\!]} = \mathrm{id}_I$ since $0_{I,I} \neq \mathrm{id}_I$ by definition of a model. Thus there exists σ' such that $\sigma \vdash \mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q \downarrow \sigma'$.

  To show the case when $b = f\!f$, we proceed as before. We then have

$$\begin{aligned}[\![c]\!]\,[\![\sigma]\!] &= [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![p]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![f\!f]\!]\,[\![\sigma]\!] \\ &= [\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,\amalg_2\,[\![\sigma]\!] = [\![q]\!]^\dagger\,\amalg_2\,[\![c_2]\!]\,[\![\sigma]\!]\end{aligned}$$

  using the fact that $[\![p]\!]\,[\![\sigma]\!] = [\![f\!f]\!]\,[\![\sigma]\!] = \amalg_2\,[\![\sigma]\!]$ and naturality of $\amalg_2$. Thus $[\![c_2]\!]\,[\![\sigma]\!]$ must be total by Lemma 3.3, which means that by induction there exists σ' such that $\sigma \vdash c_2 \downarrow \sigma'$ by a derivation $\mathcal{D}_2$, and by Lemma 6.9, $[\![c_2]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$. Continuing as before, we obtain now that

$$\begin{aligned}[\![c]\!]\,[\![\sigma]\!] &= [\![q]\!]^\dagger\,\amalg_2\,[\![c_2]\!]\,[\![\sigma]\!] = [\![q]\!]^\dagger\,\amalg_2\,[\![\sigma']\!] = (\amalg_2^\dagger\,[\![q]\!])^\dagger\,[\![\sigma']\!] = \overline{\amalg_2^\dagger\,[\![q]\!]}^\dagger\,[\![\sigma']\!] \\ &= \overline{\amalg_2^\dagger\,[\![q]\!]}\,[\![\sigma']\!] = [\![\sigma']\!]\,\overline{\amalg_2^\dagger\,[\![q]\!]\,[\![\sigma']\!]} = [\![\sigma']\!]\,\overline{\amalg_2^\dagger\,[\![q]\!]\,[\![\sigma']\!]}\,\overline{[\![q]\!]\,[\![\sigma']\!]}\end{aligned}$$

and so $\llbracket q \rrbracket \llbracket \sigma' \rrbracket$ must be total in this case as well (by arguments analogous to the corresponding case for $b = tt$), so by Lemma 6.7 there must exist b' such that $\sigma' \vdash q \rightsquigarrow b'$ by some derivation $\mathcal{D}_3$, and $\llbracket q \rrbracket \llbracket \sigma' \rrbracket = \llbracket b' \rrbracket \llbracket \sigma' \rrbracket$ by Lemma 6.5. Again, we do a case analysis depending on the value of b'.

If $b' = tt$, we have

$$\begin{aligned}\llbracket c \rrbracket \llbracket \sigma \rrbracket &= \llbracket \sigma' \rrbracket\, \overline{\amalg_2^\dagger \llbracket q \rrbracket \llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket\, \overline{\amalg_2^\dagger \llbracket tt \rrbracket \llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket\, \overline{\amalg_2^\dagger \amalg_1 \llbracket \sigma' \rrbracket} \\ &= \llbracket \sigma' \rrbracket\, \overline{0_{\Sigma,\Sigma} \llbracket \sigma' \rrbracket} = \llbracket \sigma' \rrbracket\, \overline{0_{I,\Sigma}} = \llbracket \sigma' \rrbracket\, 0_{I,I} = 0_{I,\Sigma}\end{aligned}$$

by arguments analogous to the corresponding case where $b = tt$. This contradicts the totality of $\llbracket c \rrbracket \llbracket \sigma \rrbracket$ by $\overline{\llbracket c \rrbracket \llbracket \sigma \rrbracket} = \overline{0_{I,\Sigma}} = 0_{I,I} \neq \mathrm{id}_I$, and we get by contradiction that there exists σ' such that $\sigma \vdash \textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q \downarrow \sigma'$.

If $b' = ff$, we derive $\sigma \vdash \textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q \downarrow \sigma'$ by

$$\frac{\overset{\mathcal{D}_1}{\sigma \vdash p \rightsquigarrow ff} \qquad \overset{\mathcal{D}_2}{\sigma \vdash c_2 \downarrow \sigma'} \qquad \overset{\mathcal{D}_3}{\sigma' \vdash q \rightsquigarrow ff}}{\sigma \vdash \textbf{if } p \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ fi } q \downarrow \sigma'}\ .$$

- Case $c = \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q$.

In this case, $\llbracket c \rrbracket \llbracket \sigma \rrbracket = \llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket = \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}(\beta[p, c_1, q]) \llbracket \sigma \rrbracket$. Since this is total and $\llbracket \sigma \rrbracket : I \to \Sigma$, it follows by Lemma 6.11 that either

$$\llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket = \beta[p, c_1, q]_{11} \llbracket \sigma \rrbracket$$

or there exists $n \in \omega$ such that

$$\llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket = \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket\, .$$

If $\llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket = \beta[p, c_1, q]_{11} \llbracket \sigma \rrbracket$, we have

$$\begin{aligned}\llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket &= \beta[p, c_1, q]_{11} \llbracket \sigma \rrbracket = \amalg_1^\dagger (\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \llbracket p \rrbracket^\dagger \amalg_1 \llbracket \sigma \rrbracket \\ &= \amalg_1^\dagger (\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket\, \overline{\amalg_1^\dagger \llbracket p \rrbracket}\, \llbracket \sigma \rrbracket \\ &= \amalg_1^\dagger (\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket} \\ &= \amalg_1^\dagger (\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \llbracket \sigma \rrbracket\, \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket}\ \overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket}\ \overline{\llbracket p \rrbracket \llbracket \sigma \rrbracket}\end{aligned}$$

using the trick from previously that $\llbracket p \rrbracket^\dagger \amalg_1 = (\llbracket p \rrbracket^\dagger \amalg_1)^{\dagger\dagger} = (\amalg_1^\dagger \llbracket p \rrbracket)^\dagger = \overline{\amalg_1^\dagger \llbracket p \rrbracket}^\dagger = \overline{\amalg_1^\dagger \llbracket p \rrbracket}$, $\llbracket q \rrbracket \llbracket \sigma \rrbracket = \llbracket q \rrbracket \llbracket \sigma \rrbracket\, \overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket}$, the fourth axiom of restriction idempotents and commutativity of restriction idempotents to obtain this. It follows by the totality of $\llbracket c \rrbracket \llbracket \sigma \rrbracket$ that $\overline{\llbracket p \rrbracket \llbracket \sigma \rrbracket}$ and $\overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket}$ must be total, so $\llbracket p \rrbracket \llbracket \sigma \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma \rrbracket$ must be total as well. It then follows by Lemma 6.7 that there exist b_1 and b_2 such that $\sigma \vdash p \rightsquigarrow b_1$ and $\sigma \vdash q \rightsquigarrow b_2$ by derivations $\mathcal{D}_1$ respectively $\mathcal{D}_2$. But then it follows by Lemma 6.8 that $b_1 = b_2 = tt$, as we would otherwise have $\llbracket c \rrbracket \llbracket \sigma \rrbracket = \beta[p, c_1, q]_{11} \llbracket \sigma \rrbracket = 0_{I,\Sigma}$, contradicting totality. Thus we may derive $\sigma \vdash \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \downarrow \sigma$ by

$$\frac{\overset{\mathcal{D}_1}{\sigma \vdash p \rightsquigarrow tt} \qquad \overset{\mathcal{D}_2}{\sigma \vdash q \rightsquigarrow tt}}{\sigma \vdash \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \downarrow \sigma}.$$

On the other hand, suppose that there exists $n \in \omega$ such that

$$\llbracket c \rrbracket \llbracket \sigma \rrbracket = \llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket = \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket\, .$$

Since this is total, by Lemma 3.3 $\beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket$ is total as well, and

$$\begin{aligned}
\beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket &= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \llbracket p \rrbracket^\dagger \amalg_1 \llbracket \sigma \rrbracket = \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \overline{\amalg_1^\dagger \llbracket p \rrbracket} \llbracket \sigma \rrbracket \\
&= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \llbracket \sigma \rrbracket \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket} \\
&= \amalg_2^\dagger(\mathrm{id}_\Sigma \oplus \llbracket c_1 \rrbracket) \llbracket q \rrbracket \llbracket \sigma \rrbracket \overline{\amalg_1^\dagger \llbracket p \rrbracket \llbracket \sigma \rrbracket}\, \overline{\llbracket p \rrbracket \llbracket \sigma \rrbracket}\; \overline{\llbracket q \rrbracket \llbracket \sigma \rrbracket}
\end{aligned}$$

which follows by arguments analogous to the previous case. But then $\llbracket p \rrbracket \llbracket \sigma \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma \rrbracket$ must be total as well, so by Lemma 6.7 there exist b_1 and b_2 such that $\sigma \vdash p \rightsquigarrow b_1$ and $\sigma \vdash q \rightsquigarrow b_2$ by derivations $\mathcal{D}_1$ respectively $\mathcal{D}_2$. Since $\beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket$ is total, it further follows by Lemma 6.8 that $b_1 = tt$ and $b_2 = f\!f$, as we would otherwise have $\beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket = 0_{I,\Sigma}$. Further, $\beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket = \llbracket c_1 \rrbracket \llbracket \sigma \rrbracket$, and since this is total, by induction there exists σ' such that $\sigma \vdash c_1 \downarrow \sigma'$ by some derivation $\mathcal{D}_3$, with $\llbracket c_1 \rrbracket \llbracket \sigma \rrbracket = \llbracket \sigma' \rrbracket$ by Lemma 6.9.

To summarize, we have now that

$$\begin{aligned}
\llbracket \textbf{from } p \textbf{ loop } c_1 \textbf{ until } q \rrbracket \llbracket \sigma \rrbracket &= \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \beta[p, c_1, q]_{12} \llbracket \sigma \rrbracket \\
&= \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \llbracket c_1 \rrbracket \llbracket \sigma \rrbracket \\
&= \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \llbracket \sigma' \rrbracket
\end{aligned}$$

is total, and we have derivations $\mathcal{D}_1$ of $\sigma \vdash p \rightsquigarrow tt$, $\mathcal{D}_2$ of $\sigma \vdash q \rightsquigarrow f\!f$, and $\mathcal{D}_3$ of $\sigma \vdash c_1 \downarrow \sigma'$. To finish the proof, we show by induction on n that if $\beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \llbracket \sigma' \rrbracket$ is total for any state σ', there exists a state σ'' such that $\sigma' \vdash \underline{loop}[p, c_1, q] \downarrow \sigma''$ by some derivation $\mathcal{D}_4$.

- In the base case $n = 0$, so $\beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \llbracket \sigma' \rrbracket = \beta[p, c_1, q]_{21} \llbracket \sigma' \rrbracket$. By proof analogous to previous cases, we have that $\llbracket p \rrbracket \llbracket \sigma' \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma' \rrbracket$ must be total, so by Lemma 6.7 there exist b'_1 and b'_2 such that $\sigma' \vdash p \rightsquigarrow b'_1$ and $\sigma' \vdash q \rightsquigarrow b'_1$ by derivations $\mathcal{D}'_1$ respectively $\mathcal{D}'_2$. Further, by totality, it follows by Lemma 6.8 that we must have $b'_1 = f\!f$, $b'_2 = tt$. Thus we may produce our derivation $\mathcal{D}_4$ of $\sigma' \vdash \underline{loop}[p, c_1, q] \downarrow \sigma'$ by

$$\frac{\overset{\mathcal{D}'_1}{\sigma' \vdash p \rightsquigarrow f\!f} \qquad \overset{\mathcal{D}'_2}{\sigma' \vdash q \rightsquigarrow tt}}{\sigma' \vdash \underline{loop}[p, c, q] \downarrow \sigma'}$$

- In the inductive case, we have that

$$\beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^{n+1} \llbracket \sigma' \rrbracket = \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \beta[p, c_1, q]_{22} \llbracket \sigma' \rrbracket$$

  and since this is assumed to be total, it follows by Lemma 3.3 that $\beta[p, c_1, q]_{22} \llbracket \sigma' \rrbracket$ is total as well. Again, by argument analogous to previous cases, this implies that $\llbracket p \rrbracket \llbracket \sigma' \rrbracket$ and $\llbracket q \rrbracket \llbracket \sigma' \rrbracket$ are both total, so by Lemma 6.7 there exist b'_1 and b'_2 such that $\sigma' \vdash p \rightsquigarrow b'_1$ and $\sigma' \vdash q \rightsquigarrow b'_2$ by derivations $\mathcal{D}'_1$ respectively $\mathcal{D}'_2$. Likewise, it then follows by Lemma 6.8 that since this is total, we must have $b'_1 = b'_2 = f\!f$, and so $\beta[p, c_1, q]_{22} \llbracket \sigma' \rrbracket = \llbracket c_1 \rrbracket \llbracket \sigma' \rrbracket$, again by Lemma 6.8. Since $\llbracket c_1 \rrbracket \llbracket \sigma' \rrbracket$ is total, by the outer induction hypothesis there exists a derivation $\mathcal{D}'_3$ of $\sigma' \vdash c_1 \downarrow \sigma''$ for some σ'', and so $\llbracket c_1 \rrbracket \llbracket \sigma' \rrbracket = \llbracket \sigma'' \rrbracket$ by Lemma 6.9. But then

$$\begin{aligned}
\beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^{n+1} \llbracket \sigma' \rrbracket &= \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \beta[p, c_1, q]_{22} \llbracket \sigma' \rrbracket \\
&= \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \llbracket c_1 \rrbracket \llbracket \sigma' \rrbracket \\
&= \beta[p, c_1, q]_{21} \beta[p, c_1, q]_{22}^n \llbracket \sigma'' \rrbracket
\end{aligned}$$

since this is total, by (inner) induction there exists a derivation $\mathcal{D}_4'$ of $\sigma'' \vdash \underline{loop}[p, c_1, q] \downarrow \sigma'''$. Thus, we may produce our derivation $\mathcal{D}_4$ as

$$\frac{\overset{\mathcal{D}_1'}{\sigma' \vdash p \rightsquigarrow \mathit{ff}} \qquad \overset{\mathcal{D}_2'}{\sigma' \vdash q \rightsquigarrow \mathit{ff}} \qquad \overset{\mathcal{D}_3'}{\sigma' \vdash c_1 \downarrow \sigma''} \qquad \overset{\mathcal{D}_4'}{\sigma'' \vdash \underline{loop}[p, c, q] \downarrow \sigma'''}}{\sigma' \vdash \underline{loop}[p, c_1, q] \downarrow \sigma'''}$$

concluding the internal lemma.

Since this is the case, we may finally show $\sigma \vdash \textbf{from } p \textbf{ loop } q \textbf{ until } c \downarrow \sigma'$ by

$$\frac{\overset{\mathcal{D}_1}{\sigma \vdash p \rightsquigarrow \mathit{tt}} \qquad \overset{\mathcal{D}_2}{\sigma \vdash q \rightsquigarrow \mathit{ff}} \qquad \overset{\mathcal{D}_3}{\sigma \vdash c \downarrow \sigma'} \qquad \overset{\mathcal{D}_4}{\sigma' \vdash \underline{loop}[p, c, q] \downarrow \sigma''}}{\sigma \vdash \textbf{from } p \textbf{ loop } c \textbf{ until } q \downarrow \sigma''},$$

which concludes the proof. □

7. Full abstraction

Where computational soundness and adequacy state an agreement in the notions of convergence between the operational and denotational semantics, full abstraction deals with their respective notions of equivalence (see, *e.g.*, [32, 29]). Unlike the case for computational soundness and computational adequacy, where defining a proper notion of convergence required more work on the categorical side, the tables have turned when it comes to program equivalence. In the categorical semantics, program equivalence is clear – equality of interpretations. Operationally, however, there is nothing immediately corresponding to equality of behaviour at runtime.

To produce this, we consider how two programs may behave when executed from the same start state. If they always produce the same result, we say that they are operationally equivalent. Formally, we define this as follows:

Definition 7.1. Say that programs p_1 and p_2 are *operationally equivalent*, denoted $p_1 \approx p_2$, if for all states σ, $\sigma \vdash p_1 \downarrow \sigma'$ if and only if $\sigma \vdash p_2 \downarrow \sigma'$.

A model is said to be *equationally fully abstract* (see, *e.g.*, [32]) if these two notions of program equivalence are in agreement. In the present section, we will show a sufficient condition for equational full abstraction of models of structured reversible flowchart languages. This condition will be that the given model additionally has the properties of being *I-well pointed* and *bijective on states.*

Definition 7.2. Say that a model of a structured reversible flowchart language is *I-well pointed* if, for all parallel morphisms $f, g : A \to B$, $f = g$ precisely when $fp = gp$ for all $p : I \to A$.

Definition 7.3. Say that a model $\mathscr{C}$ of a structured reversible flowchart language $\mathcal{L}$ is *bijective on states* if there is a bijective correspondence between states of $\mathcal{L}$ and *total* morphisms $I \to \Sigma$ of $\mathscr{C}$.

If a computationally sound and adequate model of a structured reversible flowchart language is bijective on states, we can show a stronger version of Lemma 6.9.

Lemma 7.4. *If $\mathscr{C}$ be a sound and adequate model of a structured reversible flowchart language which is bijective on states. Then $\sigma \vdash c \downarrow \sigma'$ iff $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$ and this is total.*

Proof. By Lemma 6.9 and Theorem 6.10, we only need to show that $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$ implies $\sigma \vdash c \downarrow \sigma'$. Assume that $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma']\!]$ and that this is total. By Theorem 6.12, there exists σ'' such that $\sigma \vdash c \downarrow \sigma''$, so by Lemma 6.9, $[\![c]\!]\,[\![\sigma]\!] = [\![\sigma'']\!]$. Thus $[\![\sigma']\!] = [\![c]\!]\,[\![\sigma]\!] = [\![\sigma'']\!]$, so $\sigma' = \sigma''$ by bijectivity on states. □

With this, we can show equational full abstraction.

Theorem 7.5 (Equational full abstraction)**.** *Let $\mathscr{C}$ be a sound and adequate model of a structured reversible flowchart language that is furthermore I-well pointed and bijective on states. Then $\mathscr{C}$ is equationally fully abstract,* i.e., *$p_1 \approx p_2$ if and only if $[\![p_1]\!] = [\![p_2]\!]$.*

Proof. Suppose $p_1 \approx p_2$, *i.e.*, for all σ, $\sigma \vdash p_1 \downarrow \sigma'$ if and only if $\sigma \vdash p_2 \downarrow \sigma'$. Let $s : I \to \Sigma$ be a morphism. Since $\mathscr{C}$ is a model of a structured reversible flowchart language, it follows that either $\overline{s} = \mathrm{id}_I$ or $\overline{s} = 0_{I,I}$ where $\mathrm{id}_I \neq 0_{I,I}$.

If $\overline{s} = 0_{I,I}$, we have $[\![p_1]\!]\,s = [\![p_2]\!]\,s = 0_{I,\Sigma}$ by unicity of the zero map.

On the other hand, if $\overline{s} = \mathrm{id}_I$, by bijectivity on states there exists σ_0 such that $s = [\![\sigma_0]\!]$. Consider now $[\![p_1]\!]\,s = [\![p_1]\!]\,[\![\sigma_0]\!]$. If this is total, by Theorem 6.12 there exists σ_0' such that $\sigma_0 \vdash p_1 \downarrow \sigma_0'$, and by $p_1 \approx p_2$, $\sigma_0 \vdash p_2 \downarrow \sigma_0'$ as well. But then, applying Lemma 6.9 on both yields that

$$[\![p_1]\!]\,s = [\![p_1]\!]\,[\![\sigma_0]\!] = [\![\sigma_0']\!] = [\![p_2]\!]\,[\![\sigma_0]\!] = [\![p_2]\!]\,s\ .$$

If, on the other hand, $[\![p_1]\!]\,s$ is not total, by the contrapositive to Theorem 6.10, there exists *no* σ_0' such that $\sigma_0 \vdash p_1 \downarrow \sigma_0'$, so by $p_1 \approx p_2$ there exists no σ_0'' such that $\sigma_0 \vdash p_1 \downarrow \sigma_0''$. But then, by the contrapositive of Theorem 6.12 and the fact that restriction idempotents on I are either id_I or $0_{I,I}$, it follows that

$$[\![p_1]\!]\,s = [\![p_1]\!]\,[\![\sigma_0]\!] = 0_{I,\Sigma} = [\![p_2]\!]\,[\![\sigma_0]\!] = [\![p_2]\!]\,s.$$

Since s was chosen arbitrarily and $[\![p_1]\!]\,s = [\![p_2]\!]\,s$ in all cases, it follows by I-well pointedness that $[\![p_1]\!] = [\![p_2]\!]$.

In the other direction, suppose $[\![p_1]\!] = [\![p_2]\!]$, let σ_0 be a state, and suppose that there exists σ_0' such that $\sigma_0 \vdash p_1 \downarrow \sigma_0'$. By Lemma 6.9, $[\![p_1]\!]\,[\![\sigma_0]\!] = [\![\sigma_0']\!]$, and by Theorem 6.10 this is total. But then, by $[\![p_1]\!] = [\![p_2]\!]$, $[\![p_2]\!]\,[\![\sigma_0]\!] = [\![p_1]\!]\,[\![\sigma_0]\!] = [\![\sigma_0']\!]$, so by Lemma 7.4, $\sigma_0 \vdash p_2 \downarrow \sigma_0'$. The other direction follows similarly. □

8. Applications

In this section, we briefly cover some applications of the developed theory: We show how the usual program invertion rules can be verified using the semantics; introduce a small reversible flowchart language, and use the results from the previous sections to give it semantics; and discuss how decisions may be used as a programming technique to naturally represent predicates in a reversible functional language.

8.1. **Verifying program inversion.** A desirable syntactic property for reversible programming languages is to be closed under program inversion, in the sense that for each program p, there is another program $\mathcal{I}[p]$ such that $[\![\mathcal{I}[p]]\!] = [\![p]\!]^\dagger$. Janus, `R-WHILE`, and `R-CORE` [40, 19, 20] are all examples of reversible programming languages with this property.

This is typically witnessed by a *program inverter* [1, 2], that is, a procedure mapping the program text of a program to the program text of its inverse program[3].

Program inversion of irreversible languages is inherently difficult because of their backwards nondeterminism and the elimination of nondeterminism requires advanced machinery (*e.g.*, LR-based parsing methods [18]). Constructing program inversion for reversible languages is typically straightforward because of their backward determinism, but exploting this object language property, *e.g.*, for proving program inverters correct, is severly hindered by the fact that conventional metalanguages do not naturally capture these object-language properties. On the other hand, the categorical foundation considered here leads to a very concise theorem regarding program inversion for all reversible flowchart languages.

Suppose that we are given a language where elementary operations are closed under program inversion (*i.e.*, where each elementary operation b has an inverse $\mathcal{I}[b]$ such that $[\![\mathcal{I}[b]]\!] = [\![b]\!]^\dagger$). We can use the semantics to verify the usual program inversion rules for **skip**, sequencing, reversible conditionals and loops as follows, by structural induction on c with the hypothesis that $[\![\mathcal{I}[c]]\!] = [\![c]\!]^\dagger$. For **skip**, we have

$$[\![\mathbf{skip}]\!]^\dagger = \mathrm{id}_\Sigma^\dagger = \mathrm{id}_\Sigma = [\![\mathbf{skip}]\!]$$

thus justifying the inversion rule $\mathcal{I}[\mathbf{skip}] = \mathbf{skip}$. Likewise for sequences,

$$[\![c_1 \ ; c_2]\!]^\dagger = ([\![c_2]\!]\,[\![c_1]\!])^\dagger = [\![c_1]\!]^\dagger\,[\![c_2]\!]^\dagger = [\![\mathcal{I}[c_1]]\!]\,[\![\mathcal{I}[c_2]]\!] = [\![\mathcal{I}[c_2] \ ; \mathcal{I}[c_1]]\!]$$

which verifies the inversion rule

$$\mathcal{I}[c_1 \ ; c_2] = \mathcal{I}[c_2] \ ; \mathcal{I}[c_1] \ \ .$$

Our approach becomes more interesting when we come to conditionals. Given some conditional statement **if** p **then** c_1 **else** c_2 **fi** q, we notice that

$$\begin{aligned}
[\![\mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q]\!]^\dagger &= ([\![q]\!]^\dagger\,([\![c_1]\!] \oplus [\![c_2]\!])\,[\![p]\!])^\dagger \\
&= [\![p]\!]^\dagger\,([\![c_1]\!]^\dagger \oplus [\![c_2]\!]^\dagger)\,[\![q]\!]^{\dagger\dagger} \\
&= [\![p]\!]^\dagger\,([\![c_1]\!]^\dagger \oplus [\![c_2]\!]^\dagger)\,[\![q]\!] \\
&= [\![p]\!]^\dagger\,([\![\mathcal{I}[c_1]]\!] \oplus [\![\mathcal{I}[c_2]]\!])\,[\![q]\!] \\
&= [\![\mathbf{if}\ q\ \mathbf{then}\ \mathcal{I}[c_1]\ \mathbf{else}\ \mathcal{I}[c_2]\ \mathbf{fi}\ p]\!]
\end{aligned}$$

which verifies the correctness of usual the inversion rule (see, *e.g.*, [19, 20])

$$\mathcal{I}[\mathbf{if}\ p\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ q] = \mathbf{if}\ q\ \mathbf{then}\ \mathcal{I}[c_1]\ \mathbf{else}\ \mathcal{I}[c_2]\ \mathbf{fi}\ p.$$

[3]While semantic inverses *are* unique, their program texts generally are not. As such, a programming language may have many different sound and complete program inverters, though they will all be equivalent up to program semantics.

Finally, for reversible loops, we have

$$\begin{aligned}
[\![\textbf{from } p \textbf{ loop } c \textbf{ until } q]\!]^\dagger &= \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}((\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger)^\dagger \\
&= \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}(((\mathrm{id}_\Sigma \oplus [\![c]\!])\,[\![q]\!]\,[\![p]\!]^\dagger)^\dagger) \\
&= \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}([\![p]\!]\,[\![q]\!]^\dagger\,(\mathrm{id}_\Sigma \oplus [\![c]\!]^\dagger)) \\
&= \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}((\mathrm{id}_\Sigma \oplus [\![c]\!]^\dagger)\,[\![p]\!]\,[\![q]\!]^\dagger) \\
&= \mathrm{Tr}^{\Sigma}_{\Sigma,\Sigma}((\mathrm{id}_\Sigma \oplus [\![\mathcal{I}[c]]\!])\,[\![p]\!]\,[\![q]\!]^\dagger) \\
&= [\![\textbf{from } q \textbf{ loop } \mathcal{I}[c] \textbf{ until } p]\!]
\end{aligned}$$

where the fact that it is a †-trace allows us to move the dagger inside the trace, and dinaturality of the trace in the second component allows us to move $\mathrm{id}_\Sigma \oplus [\![c]\!]^\dagger$ from the very right to the very left. This brief argument verifies the correctness of the inversion rule (see [20])

$$\mathcal{I}[\textbf{from } p \textbf{ loop } c \textbf{ until } q] = \textbf{from } q \textbf{ loop } \mathcal{I}[c] \textbf{ until } p$$

We summarize this in the following theorem:

Theorem 8.1. *If a reversible structured flowchart language is syntactically closed under inversion of elementary operations, it is also closed under inversion of reversible sequencing, conditionals, and loops.*

8.2. **Example: A reversible flowchart language.** Consider the following family of (neither particularly useful nor particularly useless) reversible flowchart languages for reversible computing with integer data, $\mathtt{RINT}_k$. $\mathtt{RINT}_k$ has precisely k variables available for storage, denoted $\mathtt{x_1}$ through $\mathtt{x_k}$ (of which $\mathtt{x_1}$ is designated by convention as the input/output variable), and its only atomic operations are addition and subtraction of variables, as well as addition with a constant. Variables are used as elementary predicates, with zero designating truth and non-zero values all designating falsehood. For control structures we have reversible conditionals and loops, and sequencing as usual. This gives the syntax:

$$\begin{aligned}
p &::= tt \mid \mathit{ff} \mid \mathtt{x_i} \mid p \textbf{ and } p \mid \textbf{not } p && \text{(Tests)} \\
c &::= c \,;\, c \mid \mathtt{x_i} \mathrel{+}= \mathtt{x_j} \mid \mathtt{x_i} \mathrel{-}= \mathtt{x_j} \mid \mathtt{x_i} \mathrel{+}= \overline{n} \\
&\quad \mid \textbf{if } p \textbf{ then } c \textbf{ else } c \textbf{ fi } p \\
&\quad \mid \textbf{from } p \textbf{ loop } c \textbf{ until } p && \text{(Commands)}
\end{aligned}$$

Here, $\overline{n}$ is the syntactic representation of an integer n. In the cases for addition and subtraction, we impose the additional syntactic constraints that $1 \le i \le k$, $1 \le j \le k$, and $i \ne j$, the latter to guarantee reversibility. Subtraction by a constant is not included as it may be derived straightforwardly from addition with a constant. A program in $\mathtt{RINT}_k$ is then simply a command.

We may now give semantics to this language in our framework. For a concrete model, we select the category **PInj** of sets and partial injections, which is a join inverse category with a join-preserving disjointness tensor (given on objects by the disjoint union of sets), so it is extensive in the sense of Definition 4.3 by Theorem 4.5. By our developments previously in this section, to give a full semantics to $\mathtt{RINT}_k$ in **PInj**, it suffices to provide an object (*i.e.*, a set) of stores Σ, denotations of our three classes of elementary operations (addition by a variable, addition by a constant, and subtraction by a variable) as morphisms (*i.e.*,

$$\Sigma = \mathbb{Z}^k$$
$$[\![\mathtt{x_i}]\!]\,(a_1,\dots,a_k) = \begin{cases} \amalg_1(a_1,\dots,a_k) & \text{if } a_i = 0 \\ \amalg_2(a_1,\dots,a_k) & \text{otherwise} \end{cases}$$
$$[\![\mathtt{x_i} \mathrel{+}= \mathtt{x_j}]\!]\,(a_1,\dots,a_k) = (a_1,\dots,a_{i-1},a_i+a_j,\dots,a_k)$$
$$[\![\mathtt{x_i} \mathrel{+}= \overline{n}]\!]\,(a_1,\dots,a_k) = (a_1,\dots,a_{i-1},a_i+n,\dots,a_k)$$
$$[\![\mathtt{x_i} \mathrel{-}= \mathtt{x_j}]\!]\,(a_1,\dots,a_k) = (a_1,\dots,a_{i-1},a_i-a_j,\dots,a_k)$$

Figure 4: The object of stores and semantics of elementary operations and predicates of $\mathtt{RINT}_k$ in **PInj**.

partial injective functions) $\Sigma \to \Sigma$, and denotations of our class of elementary predicates (here, testing whether a variable is zero or not) as decisions $\Sigma \to \Sigma \oplus \Sigma$. These are all shown in Figure 4. It is uncomplicated to show that all of these are partial injective functions, and that the denotation of each predicate $[\![\mathtt{x_i}]\!]$ is a decision, so that this is, in fact, a model of $\mathtt{RINT}_k$ in **PInj**.

We can now reap the benefits in the form of a reversibility theorem for free. Operationally, semantic reversibility of a program is taken to mean that it is *locally* forward and backward deterministic; roughly, that any execution of any subpart of that program is performed in a way that is both forward and backward deterministic (see, *e.g.*, [6]). Denotationally (see [25, Sec. 1.1]), the semantic reversibility of a program becomes the property that all of its meaningful subprograms (including itself) have denotations as partial isomorphisms. Since every meaningful $\mathtt{RINT}_k$ program fragment takes its semantics in an inverse category, reversibility follows directly.

Theorem 8.2 (Reversibility). *Every $\mathtt{RINT}_k$ program is semantically reversible.*

Further, since we can straightforwardly show that $[\![\mathtt{x_i} \mathrel{+}= \mathtt{x_j}]\!]^\dagger = [\![\mathtt{x_i} \mathrel{-}= \mathtt{x_j}]\!]$ and $[\![\mathtt{x_i} \mathrel{+}= \overline{n}]\!]^\dagger = [\![\mathtt{x_i} \mathrel{+}= \overline{-n}\,]\!]$, we can use the technique from Sec. 8.1 to obtain a sound and complete program inverter.

Theorem 8.3 (Program inversion). *$\mathtt{RINT}_k$ has a (sound and complete) program inverter. In particular, for every $\mathtt{RINT}_k$ program p there exists a program $\mathcal{I}[p]$ such that $[\![\mathcal{I}[p]]\!] = [\![p]\!]^\dagger$.*

8.3. **Decisions as a programming technique.** Decisions offer a solution to the awkwardness in representing predicates reversibly, as the usual representations of predicates on X as maps $X \to \mathit{Bool}$ fail to be reversible in all but the most trivial cases. On the programming side, the reversible *duplication/equality operator* $\lfloor \cdot \rfloor$ (see [17, 38]), defined on lists as

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

can be seen as a distant ancestor to this idea of predicates as decisions, in that it provides an ad-hoc solution to the problem of checking whether two values are equal in a reversible manner.

$$\begin{aligned} &\mathit{pnot} && :: \mathit{PBool}\,\alpha \leftrightarrow \mathit{PBool}\,\alpha \\ &\mathit{pnot}\,(\mathit{True}\ x) && = \mathit{False}\ x \\ &\mathit{pnot}\,(\mathit{False}\ x) && = \mathit{True}\ x \end{aligned} \qquad \begin{aligned} &\mathit{peven} && :: \mathit{Nat} \leftrightarrow \mathit{PBool}\,\mathit{Nat} \\ &\mathit{peven}\,0 && = \mathit{True}\,0 \\ &\mathit{peven}\,(n+1) && = \mathit{fmap}\,(+1)\,(\mathit{pnot}\,(\mathit{peven}\ n)) \end{aligned}$$

Figure 5: The definition of the *even*-predicate as a decision on natural numbers.

Decisions offer a more systematic approach: They suggest that one ought to define Boolean values in reversible functional programming not in the usual way, but rather by means of the polymorphic datatype

$$\mathbf{data}\ \mathit{PBool}\,\alpha\ =\ \mathit{True}\,\alpha\ \mid\ \mathit{False}\,\alpha$$

storing not only the *result*, but also *what* was tested to begin with. With this definition, negation on these polymorphic Booleans (*pnot*) may be defined straightforwardly as shown in Figure 5. In turn, this allows for more complex predicates to be expressed in a largely familiar way. For example, the decision for testing whether a natural number is even (*peven*) is also shown in Figure 5, with *fmap* given in the straightforward way on polymorphic Booleans, *i.e.*

$$\begin{aligned} &\mathit{fmap} && :: (\alpha \leftrightarrow \beta) \rightarrow (\mathit{PBool}\,\alpha \leftrightarrow \mathit{PBool}\,\beta) \\ &\mathit{fmap}\ f\ (\mathit{True}\ x) && = \mathit{True}\ (f\ x) \\ &\mathit{fmap}\ f\ (\mathit{False}\ x) && = \mathit{False}\ (f\ x) \quad . \end{aligned}$$

For comparison with the *peven* function shown in Figure 5, the corresponding irreversible predicate is typically defined as follows, with *not* the usual negation of Booleans

$$\begin{aligned} &\mathit{even} && :: \mathit{Nat} \rightarrow \mathit{Bool} \\ &\mathit{even}\,0 && = \mathit{True} \\ &\mathit{even}\,(n+1) && = \mathit{not}\,(\mathit{even}\ n) \quad . \end{aligned}$$

As such, the reversible implementation as a decision is nearly identical, the only difference being the use of *fmap* in the definition of *peven* to recover the input value once the branch has been decided.

9. Concluding remarks

In the present paper, we have built on the work on extensive restriction categories [10, 11, 12] to derive a related concept of extensivity for inverse categories. We have used this concept to give a novel reversible representation of predicates and their corresponding assertions in (specifically extensive) join inverse categories with a disjointness tensor, and in turn used these to model the fundamental control structures of reversible loops and conditionals in structured reversible flowchart languages. We have shown that these categorical semantics are computationally sound and adequate with respect to the operational semantics, and given a sufficient condition for equational full abstraction.

Further, this approach also allowed us to derive a program inversion theorem for structured reversible flowchart languages, and we illustrated our approach by developing a family of structured reversible flowchart languages and using our framework to give it denotational semantics, with theorems regarding reversibility and program inversion for free.

The idea to represent predicates by decisions was inspired by the *instruments* associated with predicates in Effectus theory [23]. Given that *side effect free* instruments ι in Effectus theory satisfy a similar rule as decisions in extensive restriction categories, namely $\nabla\iota = \mathrm{id}$, and that Boolean effecti are extensive, it could be interesting to explore the connections between extensive restriction categories and Boolean effecti, especially as regards their internal logic.

Finally, on the programming language side, it could be interesting to further explore how decisions can be used in reversible programming, *e.g.*, to do the heavy lifting involved in pattern matching and branch joining. As our focus has been on the representation of predicates, our approach may be easily adapted to other reversible flowchart structures, *e.g.*, Janus-style loops [40].

References

[1] Abramov, S. M. and R. Glück, *The universal resolving algorithm: inverse computation in a functional language*, in: R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction. Proceedings*, Lecture Notes in Computer Science **1837** (2000), pp. 187–212.

[2] Abramov, S. M. and R. Glück, *The universal resolving algorithm and its correctness: inverse computation in a functional language*, Science of Computer Programming **43** (2002), pp. 193–229.

[3] Axelsen, H. B. and R. Glück, *What do reversible programs compute?*, in: M. Hofmann, editor, *Foundations of Software Science and Computation Structures. Proceedings*, Lecture Notes in Computer Science **6604** (2011), pp. 42–56.

[4] Axelsen, H. B., R. Glück and T. Yokoyama, *Reversible machine code and its abstract processor architecture*, in: V. Diekert, M. V. Volkov and A. Voronkov, editors, *Computer Science – Theory and Applications. Proceedings*, Lecture Notes in Computer Science **4649** (2007), pp. 56–69.

[5] Axelsen, H. B. and R. Kaarsgaard, *Join inverse categories as models of reversible recursion*, in: B. Jacobs and C. Löding, editors, *FOSSACS 2016, Proceedings*, number 9634 in Lecture Notes in Computer Science (2016), pp. 73–90.

[6] Bennett, C. H., *Logical reversibility of computation*, IBM Journal of Research and Development **17** (1973), pp. 525–532.

[7] Carboni, A., S. Lack and R. F. C. Walters, *Introduction to extensive and distributive categories*, Journal of Pure and Applied Algebra **84** (1993), pp. 145 – 158.

[8] Carothers, C. D., K. S. Perumalla and R. M. Fujimoto, *Efficient optimistic parallel simulations using reverse computation*, ACM Trans. Model. Comput. Simul. **9** (1999), pp. 224–253.

[9] Cockett, J. R. B., *Itegories & PCAs* (2007), slides from talk at FMCS 2007.

[10] Cockett, J. R. B. and S. Lack, *Restriction categories I: Categories of partial maps*, Theoretical Computer Science **270** (2002), pp. 223–259.

[11] Cockett, J. R. B. and S. Lack, *Restriction categories II: Partial map classification*, Theoretical Computer Science **294** (2003), pp. 61–102.

[12] Cockett, R. and S. Lack, *Restriction categories III: Colimits, partial limits and extensivity*, Mathematical Structures in Computer Science **17** (2007), pp. 775–817.

[13] Fiore, M. P., "Axiomatic Domain Theory in Categories of Partial Maps," Ph.D. thesis, University of Edinburgh (1994).

[14] Frank, M. P., "Reversibility for efficient computing," Ph.D. thesis, EECS Dept., Massachusetts Institute of Technology (1999).

[15] Giles, B. G., "An Investigation of some Theoretical Aspects of Reversible Computing," Ph.D. thesis, University of Calgary (2014).

[16] Glück, R. and R. Kaarsgaard, *A categorical foundation for structured reversible flowchart languages*, in: A. Silva, editor, *Proceedings of the 33rd Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII)*, Electronic Notes in Theoretical Computer Science **336** (2017), pp. 155–171.

[17] Glück, R. and M. Kawabe, *A program inverter for a functional language with equality and constructors*, in: A. Ohori, editor, *Programming Languages and Systems. Proceedings*, Lecture Notes in Computer Science **2895** (2003), pp. 246–264.

[18] Glück, R. and M. Kawabe, *A method for automatic program inversion based on LR(0) parsing*, Fundamenta Informaticae **66** (2005), pp. 367–395.

[19] Glück, R. and T. Yokoyama, *A linear-time self-interpreter of a reversible imperative language*, Computer Software **33** (2016), pp. 108–128.

[20] Glück, R. and T. Yokoyama, *A minimalist's reversible while language*, IEICE Transactions on Information and Systems **E100-D** (2017), pp. 1026–1034.

[21] Guo, X., "Products, Joins, Meets, and Ranges in Restriction Categories," Ph.D. thesis, University of Calgary (2012).

[22] Haulund, T., T. Mogensen and R. Glück, *Implementing reversible object-oriented language features on reversible machines*, in: I. Phillips and H. Rahaman, editors, *Reversible Computation. Proceedings*, Lecture Notes in Computer Science **10301**, 2017, pp. 66–73.

[23] Jacobs, B., *New directions in categorical logic, for classical, probabilistic and quantum logic*, Logical Methods in Computer Science **11** (2015), pp. 1–76.

[24] Joyal, A., R. Street and D. Verity, *Traced monoidal categories*, Mathematical Proceedings of the Cambridge Philosophical Society **119** (1996), pp. 447–468.

[25] Kaarsgaard, R., "The Logic of Reversible Computing – Theory and Practice," Ph.D. thesis, University of Copenhagen (2018).

[26] Kaarsgaard, R., H. B. Axelsen and R. Glück, *Join inverse categories and reversible recursion*, Journal of Logical and Algebraic Methods in Programming **87** (2017), pp. 33–50.

[27] Kastl, J., *Inverse categories*, in: H.-J. Hoehnke, editor, *Algebraische Modelle, Kategorien und Gruppoide*, Studien zur Algebra und ihre Anwendungen **7**, Akademie-Verlag, 1979 pp. 51–60.

[28] Mogensen, T. Æ., *Partial evaluation of the reversible language Janus*, in: *Partial Evaluation and Program Manipulation. Proceedings* (2011), pp. 23–32.

[29] Plotkin, G., *Full abstraction, totality and PCF*, Mathematical Structures in Computer Science **9** (1999), pp. 1–20.

[30] Selinger, P., *Dagger compact closed categories and completely positive maps*, Electronic Notes in Theoretical Computer Science **170** (2007), pp. 139–163.

[31] Selinger, P., *A survey of graphical languages for monoidal categories*, in: B. Coecke, editor, *New Structures for Physics*, Lecture Notes in Physics **813** (2011), pp. 289–355.

[32] Stoughton, A., "Fully Abstract Models of Programming Languages," Pitman Publishing, Inc., 1988.

[33] Thomsen, M. K., H. B. Axelsen and R. Glück, *A reversible processor architecture and its reversible logic design*, in: A. De Vos and R. Wille, editors, *Reversible Computation. Proceedings*, Lecture Notes in Computer Science **7165** (2012), pp. 30–42.

[34] Thomsen, M. K., R. Glück and H. B. Axelsen, *Reversible arithmetic logic unit for quantum arithmetic*, Journal of Physics A: Mathematical and Theoretical **43** (2010), p. 382002.

[35] Winskel, G., "The Formal Semantics of Programming Languages: An Introduction," MIT Press, 1993.

[36] Yokoyama, T., H. B. Axelsen and R. Glück, *Reversible flowchart languages and the structured reversible program theorem*, in: L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir and I. Walukiewicz, editors, *International Colloquium on Automata, Languages and Programming. Proceedings*, Lecture Notes in Computer Science **5126** (2008), pp. 258–270.

[37] Yokoyama, T., H. B. Axelsen and R. Glück, *Optimizing reversible simulation of injective functions*, Journal of Multiple-Valued Logic and Soft Computing **18** (2012), pp. 5–24.

[38] Yokoyama, T., H. B. Axelsen and R. Glück, *Towards a reversible functional language*, in: A. De Vos and R. Wille, editors, *RC 2011*, Lecture Notes in Computer Science **7165** (2012), pp. 14–29.

[39] Yokoyama, T., H. B. Axelsen and R. Glück, *Fundamentals of reversible flowchart languages*, Theoretical Computer Science **611** (2016), pp. 87–115.

[40] Yokoyama, T. and R. Glück, *A reversible programming language and its invertible self-interpreter*, in: *Partial Evaluation and Semantics-Based Program Manipulation. Proceedings* (2007), pp. 144–153.