Program Logics via Distributive Monoidal Categories

ANONYMOUS AUTHOR(S)

We derive multiple program logics, including correctness, incorrectness, and relational Hoare logic, from the axioms of imperative categories: uniformly traced distributive copy-discard categories. We introduce an internal language for imperative multicategories, on top of which we derive combinators for an adaptation of Dijkstra's guarded command language. Rules of program logics are derived from this internal language.

ACM Reference Format:
Anonymous Author(s). 2025. Program Logics via Distributive Monoidal Categories. 1, 1 (July 2025), 52 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Program logics are sets of derivation rules used to reason about program behaviour under input and output conditions. Statements are written as triples $\{p\}\ c\ \{q\}$ of a command c, a precondition p and a postcondition q. The semantics of such a triple, though, depends on the behaviour one is interested in studying. For program correctness, intuitively, the triple is valid if, starting on input states that satisfy p, the output states of the program satisfy q. For example, the following rule of Hoare logic derives a correctness triple for a loop from the correctness triple of its body.

$$\frac{\{b \wedge p\}\ c\ \{p\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{(\neg b) \wedge p\}} \tag{1}$$

However, correctness is only one of the possible triple interpretations; intensive research has produced logics for a myriad of triple interpretations, and for multiple program semantics.

Program logics start by fixing a semantics for their commands, an interpretation for their triples, and derivation rules for its logic. Command semantics can be partial [Hoa69, Ben04], relational [Win93, O'H19] or stochastic [Kam18, BKOZB12, ZDS23]. Triples can capture program correctness [Hoa69], incorrectness [dVK11, O'H19] or quantitative aspects of execution [ZDS23, ABDG25]. After these two choices, the logic is completed with a set of derivation rules that capture the relevant behaviour and are sound for the intended semantics. While they appear to follow some general pattern, the rules of program logics are defined on a case-by-case basis.

We propose the algebraic structure of imperative categories—a variant of *Elgot distributive categories*—as a foundation for program logics. From the axioms of imperative categories, we derive the usual rules of various program logics. From the models of imperative categories, we expand the scope of these rules beyond a fixed semantics. The categorical structure becomes common to the usual relational, partial, and probabilistic semantics, while remaining more general.

Imperative categories come with an internal language that we develop and employ through the paper: an internal language that mimics *unstructured programming*, with arbitrary jumps to labelled looping points (marked by "**`loop`**" followed by a label). Unstructured programming is needed for full expressivity, but certainly not always desirable [Dij68]; in fact, while unstructured and typed,

ACM XXXX-XXXX/2025/7-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the internal language is actually inspired by a structured and untyped one: the famous Dijkstra's *guarded command language* [Dij75].

Dijkstra's *command language* is recovered from the endomorphisms of imperative categories. The simplest command combinators of the language—skip and concatenation (;)—feature as the identity and endomorphism composition. Choice and iteration (if-then-else and while) feature as a cocartesian and traced monoidal structure. All command combinators are derivable from the unstructured internal language; for instance, if-then-else and while are defined in these terms.

$$\text{if } b \text{ then } c_1 \text{ else } c_2 \equiv b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus c_1, c_2]; \tag{2}$$

$$\text{while } b \text{ do } c \equiv \mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{x}.\ b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus c[\boldsymbol{\eta} \setminus \vec{x}.\boldsymbol{\alpha}(\vec{x})],\ \vec{x}.\boldsymbol{\eta}(\vec{x})]\}; \tag{3}$$

These read as follow: to execute "if b then c_1 else c_2", execute b but replace each of its two exit conditions ($\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$) by the two branches (c_1 and c_2); to execute "while b do c", start by labelling a looping point ($\boldsymbol{\alpha}$) and then execute b but replacing its first exit condition ($\boldsymbol{\alpha}_1$) with the body of the loop (c)—while replacing c's exit condition ($\boldsymbol{\eta}$) by the looping label—and its second exit condition with its, now only, exit condition ($\boldsymbol{\eta}$).

While less familiar, the internal language derives the usual reasoning principles: for instance, the previous definitions—together with an auxiliary skip $\equiv \boldsymbol{\eta}(\vec{x})$—imply loop unfolding (4).

$$\text{while } b \text{ do } c \equiv \text{if } b \text{ then } (c\ ;\ \text{while } b \text{ do } c) \text{ else skip}; \tag{4}$$

Around commands, notice how we pass a vector of variables ($\vec{x}$), carrying the state of loops and choices. This sort of *state-passing translation* requires a second monoidal—or premonoidal—structure, with the ability to copy and discard the value of variables. It enables variable assignment: if both x_i and x_j are variables in the vector that we pass as state, then the following command stores in x_i the value of $\boldsymbol{f}(x_j)$.

$$(x_i \coloneqq \boldsymbol{f}(x_j)) \ \equiv\ \boldsymbol{f}(x_j)\{x_i.\boldsymbol{\eta}(\vec{x})\}.$$

As a side benefit, the second monoidal structure provides the extra expressivity needed to define couplings of programs, validity in *relational Hoare triples*, and notions of totality and determinism, useful in stochastic and partial semantics.

1.1 Interpreting triples

The interpretation of program triples rests on comparing programs: the validity of a Hoare triple $\{p\}\ c\ \{q\}$ will be defined as an inequality, assert p ; $c \leq c$; assert q. In the category of relations, where morphisms are ordered by inclusion, we recover the validity of a partial correctness triple: it compares the subset c ; assert q of possible final states with the subset assert p ; c of possible outputs of c on inputs that belong to p. In general, we require a poset enrichment on imperative categories, leading to posetal imperative categories: poset-enriched categories with *(i)* traced coproducts and a second *(ii)* monoidal copy-discard structure, interacting by distributivity.

The most important axiom for this posetal structure is posetal uniformity, which justifies *loop invariants*. Intuitively, it says that if a command c is invariant under a branch guarded by b, then it remains invariant under a loop guarded by b. That is, c; $(\!|b|\!)\{c_1\}\{\text{skip}\} \leq (\!|b|\!)\{c_2; c_0\}\{c_3\}$ implies c_0; while b do $c_1 \leq$ while b do $c_2; c_3$.

With this interpretation, let us prove validity of the example triple we just introduced.

Proposition 1. *The triple in Equation* (1) *is valid when* b *is deterministic.*

Proof. We reason by *(i)* interchange of predicates and guards, *(ii)* determinism of the guard b, *(iii)* the definition of conjunction, and *(iv)* the assumption of the rule, $\{b \wedge p\}\ c\ \{p\}$.

$$\text{assert } p;\ (\!|b|\!)\{c\}\{\text{skip}\} \overset{(i)}{=}$$

$$
\begin{aligned}
&(\!|b|\!)\{\mathsf{assert}\, p; c\}\{\mathsf{assert}\, p\} &\overset{(ii)}{=}\\
&(\!|b|\!)\{\mathsf{assert}\, b^{\#}; \mathsf{assert}\, p; c\}\{\mathsf{assert}\, (\neg b)^{\#}; \mathsf{assert}\, p\} &\overset{(iii)}{=}\\
&(\!|b|\!)\{\mathsf{assert}(b^{\#} \wedge p); c\}\{\mathsf{assert}((\neg b)^{\#} \wedge p)\} &\overset{(iv)}{\leq}\\
&(\!|b|\!)\{c; \mathsf{assert}\, p\}\{\mathsf{assert}((\neg b)^{\#} \wedge p)\}.
\end{aligned}
$$

We conclude, by posetal uniformity, that $\mathsf{assert}\, p; \mathsf{while}\, b\, \mathsf{do}\, c \leq \mathsf{while}\, b\, \mathsf{do}\, c; \mathsf{assert}(\neg b \wedge p)$. This means that $\{p\}$ $\mathsf{while}\, b\, \mathsf{do}\, c$ $\{(\neg b) \wedge p\}$ is valid. □

1.2 Contributions

We introduce imperative multicategories as traced distributive copy-discard multicategories. We provide an internal language taking sound semantics in imperative categories (Theorem 54), and we prove it complete by exhibiting a syntactic model (Theorem 56). In terms of this internal language, we derive combinators for guards, predicates, commands, and states, inspired by Dijkstra's *guarded command language* (Section 3).

Finally, we classify triple shapes from various program logics (Section 5), and we prove the derivation rules for *Hoare logic*, *incorrectness logic*, and an *outcome-like logic* (Theorems 79, 81 and 83). We extend these to their relational versions, proving the derivation rules for *relational Hoare logic* and a *relational incorrectness logic* (Theorems 88 and 90).

1.3 Synopsis

Section 2 introduces an internal language for imperative multicategories and posetal imperative multicategories. Section 3 specializes the language for the elements of a generic program triple and derives a version of Dijkstra's *guarded command language*. Section 4 provides categorical denotational semantics in terms of *posetally-enriched traced distributive copy-discard multicategories*. Section 5 derives *correctness* triples, *incorrectness* triples, and *outcome-like* triples in any imperative multicategory. Section 6 derives *relational correctness* triples and *relational incorrectness* triples again from the axioms of imperative multicategories.

1.4 Related work

Categorical program semantics. Categorical program semantics has a long tradition [LS88, Ole83, Win93]. In particular, distributive categories are since long used to model both control flow and data flow of programs [Coc93, CLW93, Wal92]. More specifically, distributive monoidal categories with copy-discard structure have naturally appeared in non-deterministic, partial, and stochastic semantics [LCS25, Nes25]. The approach is compatible with the long tradition of using monads for computations [Mog91, Wad98, BK99, BHM00].

Arbib and Manes employ traced cocartesian categories to express the control flow of programs [AM80], generalising Elgot's techniques for the interpretation of iteration and choice in partial functions [Elg75]; but also apart from their work, categorical semantics for iteration has been studied extensively [BÉ93, SP00]. Of particular relevance to our work is the metalanguage for guarded iteration by Goncharov, Rauch and Schröder [GRS21]; and the recent denotational semantics of static single assingment of Ghalayini and Krishnaswami [GK24]. When reasoning about the semantics of loops, we employ Hasuo's generic trace theory [Has06, HJS06], which builds on Fiore's work on coinduction [Fio93, Fio96]. Uniform traces need not to exist in cocartesian categories. In our examples, we ensure the existence of uniform traces by relying on partially additive monads [Jac10], which ensure a form of iteration in the Kleisli category less restrictive than additive monads [CJ13] or Kleene monads [Gon10].

Categorical logic. The guarded command syntax for programs distinguishes between guards and commands. We interpret this distinction in the categorical setting following the ideas from effectus theory [Jac15], where the logic on guards derives from categorical structure.

The structure of hom-sets in imperative categories resembles that of Kleene algebras with tests [Koz97] and their probabilistic variation [MCM06], and of guarded Kleene algebras with tests [SFH⁺19, GBG25a] and their probabilistic [RKK⁺23] and approximate variants [GBG25b]. Guards in imperative categories do not in general form a boolean algebra as they are not necessarily deterministic.

Program logics. Since the work of Floyd [Flo93] and Hoare [Hoa69, CH72] on correctness assertions about programs, much work on program logics has extended the scope of the original logic. Separation logic [Rey02, ORY01] considers programs that access globally shared data, incorrectness logic considers assertions about faults of programs [dVK11, O'H19], and outcome logics [ZDS23, ZSS24, ZKST25] provide a synthesis of correctness and incorrectness reasoning. Verification of probabilistic programs is an active research area that takes another view on program logics studying weakest precondition and strongest postcondition calculi [KK17, Kam18, ZK22].

Relational program logics extend the reasoning of program logics to pairs of programs considering binary relations between their inputs instead of predicates on the inputs of one program alone. As in the predicate version, relational program logics can focus on correctness assertions about deterministic programs [Ben04], or be extended to probabilistic semantics [BGZB09, ABDG25] and approximate reasoning [Olm14, BKOZB12, Sat16, ABH⁺21].

Categorical approaches to program logics are not new. Manes and Arbib describe the control flow of Hoare logic with traced cocartesian categories [MA12]. Outcome logic considers a class of semantic universes given by Kleisli categories of monads with some extra structure [ZDS23]. Program triples can also be seen as fibrations over a category of programs [MZ15, MZ16] or as functors to monotone relations [AMMO09]. More recently, the structure of distributive categories as been shown to derive the rules of Hoare logic, restricted to the relational semantics [BDD25].

2 An internal distributive language

Program logics follow simple imperative languages—e.g. *Dijkstra's guarded command language* [Dij75]. These tend to be bad candidates for a categorical internal language: many are untyped, and many are too redundant to construct free categories. For instance, many have explicit commands for identity (skip) and composition ($\fatsemi$), implicitly blocking categorical cut-elimination; many do poorly on relevant case-matching, rendering some categorical constructions impossible.

This section introduces the formal internal language we use for the rest of the paper. Next sections will develop its semantics in terms of imperative categories.

2.1 Signatures: values, generators, and basic types

A distributive signature is a structure apt to represent all the morphisms of a distributive category without their compositional structure. Instead of nesting sums and tensors, it exploits that every nesting of sums and tensors can be normalized—not uniquely—into a sum of tensors of basic types. In other words, all the morphisms of a distributive category can be recovered from those between sums of tensors,

$$f\colon\ \textstyle\sum_{i=1}^{\ell} \bigotimes_{j=1}^{n_i} X_j^i \to \sum_{i=1}^{p} \bigotimes_{j=1}^{m_i} Y_j^i .$$

And moreover, because of the universal property of coproducts, these correspond uniquely to tuples of morphisms from a tensor of basic types into a sum of tensors of basic types,

$$\big(\, f_i\colon\ \textstyle\bigotimes_{j=1}^{n_i} X_j^i \to \sum_{i=1}^{p} \bigotimes_{j=1}^{m_i} Y_j^i \,\big)_{i=1}^{\ell}.$$

Thus, generators—the elements of a distributive signature—will be interpreted as inducing a morphism from a product, $\bigotimes_{j=1}^{n} X_j$, to a sum of products, $\sum_{i=1}^{p} \bigotimes_{j=1}^{m_i} Y_j^i$.

Definition 2 (Distributive signature)**.** A *distributive signature*, $(\mathcal{B}, \mathcal{G})$, is given by a set whose elements we call *basic types*, $\mathcal{B}$, and, for each list of basic types $\{X_i \in \mathcal{B}\}_{i=1}^{n}$, and each list of lists of basic types, $\{\{Y_j^i \in \mathcal{B}\}_{j=1}^{m_i}\}_{i=1}^{p}$, a set, $\mathcal{G}(X_1, ..., X_n; [Y_1^1, ..., Y_{m_1}^1], ..., [Y_1^\ell, ..., Y_{m_1}^\ell])$, whose elements we call *generators*.

All morphisms in a distributive category can be brought to this form: any morphism from a coproduct is determined by a tuple of generators; morphisms between non-normalized polynomials correspond bijectively morphisms between any choice of normalizations.

Remark 3. Explicit product and coproduct types will not be needed: primitive types on the language are normalized polynomials of basic types. This does not mean we cannot include them explicitly—they are sometimes convenient—but they will be derived notions: we introduce them with bijections to primitive types, constituting their introduction/elimination pair.

2.2 Language primitives

Let us state the three constructors that form the terms of the formal language that we employ for traced distributive copy-discard multicategories. The language—in the style of categorical cut-elimination [Whi41, Joy95, RC01, Shu16]—tries to be as minimalistic as possible, avoiding redundancy of constructors: ideally, every term would correspond uniquely to a morphism in a free traced distributive copy-discard multicategory without any extra quotienting. Indeed, we only use quotienting for α-equivalence and four axioms, regarding commutativity and loops (in Section 2.4).

Definition 4 (Variables, labels, contexts, and indices)**.** Let $\mathbf{V}$ be a countable infinite set whose elements we call *variables*. Let $\mathbf{A}$ be a countable infinite set whose elements we call *labels*. A *context*, $\Gamma = x_1 : X_1, ..., x_n : X_n$, is a list of variables and basic types, i.e. $\Gamma \in \mathsf{List}(\mathbf{V} \times \mathcal{B})$. *Indices* are lists of labels and contexts, i.e. $\mathsf{Idx} = \mathsf{List}(\mathbb{A} \times \mathsf{Ctx})$.

Remark 5. Labels naturally appear when reasoning about jumps in Hoare logic [CH72]; they also match the *exit conditions* of incorrectness logic [O'H19].

Axiom 6 (Primitive terms)**.** *Terms* of the internal language, over a distributive signature $(\mathcal{B}, \mathcal{G})$, are inductively generated by the following rules.

Return
$$\frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad (\boldsymbol{\alpha} \colon X_1, ..., X_n) \in \Delta}{\Gamma \vdash \boldsymbol{\alpha}(x_1, ..., x_n) : \Delta}$$

Generator
$$\frac{\begin{array}{c} f \in \mathcal{G}(X_1, ..., X_n; (Y_{1,1}, ..., Y_{1,m_1}), ..., (Y_{\ell,1}, ..., Y_{\ell,m_\ell})) \\ \{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad \{(y_{i,1} : Y_{i,1}), ..., (y_{i,m_i} : Y_{i,m_i}), \Gamma \vdash p_i : \Delta\}_{i=1}^{\ell} \end{array}}{\Gamma \vdash f(x_1, ..., x_n)\{y_{i,1}, ..., y_{i,m_i}.\, p_i\}_{i=1}^{\ell}}$$

Loop
$$\frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad (u_1 : X_1), ..., (u_n : X_n), \Gamma \vdash p : (\boldsymbol{\alpha} \colon X_1, ..., X_n), \Delta}{\Gamma \vdash \mathbf{loop}\, \boldsymbol{\alpha}(x_1, ..., x_n)\{u_1, ..., u_n.\, p\} : \Delta}$$

- The Return rule states that, given an label, $(\boldsymbol{\alpha} : X_1, ..., X_n) \in \Delta$, and a well-typed list of variables in context, $\{(x_i : X_i) \in \Gamma\}_{i=1}^{n}$, a term may just point to that label.
- The Generator rule states that, given any generator, f, with well-typed list of variables, $\{(x_i : X_i) \in \Gamma\}_{i=1}^{n}$, and a term for each one of its possible branches, $\{p_i\}_{i=1}^{\ell}$, we can evaluate the generator and branch according to its result.

- The LOOP rule states that we can introduce a label, $\boldsymbol{\alpha}(x_1, ..., x_n)$, to which the rest of the term, p, may now jump.

From now on, let us use vector notation for lists when convenient: for instance, $\vec{x} : \vec{X}$ will mean $x_1 : X_1, ..., x_n : X_n$, and $\vec{y}_i$ will mean $y^i_1, ..., y^i_{m_i}$.

Remark 7. We work up to α-equivalence of both variables and labels. While its formalization is a routine matter, the interested reader can follow Section A.1.

2.3 Substitution

Substitution appears as a derived rule: it builds terms that, while structurally similar, employ variables differently. Most derived structural rules (e.g., exchange, contraction, or weakening) will follow from substitution. In the same way that we substitute variables, we can substitute labels. The substitution rule for labels is based in the substitution rule of *clones* (or *Lawvere theories*).

Definition 8 (Variable substitution). *Substitution* of a list of variables, $\vec{u} = u_1, ..., u_n$, by a list of variables, $\vec{v} = v_1, ..., v_n$, is defined by $u_i[\vec{u} \setminus \vec{v}] = v_i$, and $w[\vec{u} \setminus \vec{v}] = w$ when $\{w \neq u_i\}_{i=1}^n$. Substitution extends inductively to terms, as follows.

$$(\boldsymbol{\alpha}(x_1, ..., x_n))[\vec{u} \setminus \vec{v}] \equiv \boldsymbol{\alpha}(x_1[\vec{u} \setminus \vec{v}], ..., x_n[\vec{u} \setminus \vec{v}]);$$
$$(\mathbf{loop}\ \boldsymbol{\alpha}(x_1, ..., x_n)\{y_1, ..., y_n.\ p\})[\vec{u} \setminus \vec{v}] \equiv \mathbf{loop}\ \boldsymbol{\alpha}(x_1, ..., x_n)\{y_1, ..., y_n.\ p[\vec{u} \setminus \vec{v}]\};$$
$$(f(x_1, ..., x_n)\{y_1, ..., y_m.\ p_i\}_i)[\vec{u} \setminus \vec{v}] \equiv f(x_1[\vec{u} \setminus \vec{v}], ..., x_n[\vec{u} \setminus \vec{v}])\{y_{i,1}, ..., y_{i,m_i}.\ p_i[\vec{u} \setminus \vec{v}]\}_i;$$

For the last two clauses, we must assume—without loss of generality, thanks to α-equivalence—that all variables that appear bound, $y_1, ..., y_n$ and $y_{i,1}, ..., y_{i,m_i}$, are fresh.

Definition 9 (Label substitution). *Substitution* of a label, $\boldsymbol{\alpha}$, by a term q with a list of bound variables $\vec{u}$, inside a term p, is inductively defined as follows.

$$\boldsymbol{\alpha}(\vec{x})[\boldsymbol{\alpha} \setminus \vec{u}.q] \equiv q[\vec{u} \setminus \vec{x}];$$
$$\boldsymbol{\omega}(\vec{x})[\boldsymbol{\alpha} \setminus \vec{u}.q] \equiv \boldsymbol{\omega}(\vec{x}),\ \text{when } \boldsymbol{\omega} \neq \boldsymbol{\alpha};$$
$$(\mathbf{loop}\ \boldsymbol{\beta}(\vec{x})\{\vec{y}.\ p\})[\boldsymbol{\alpha} \setminus \vec{u}.q] \equiv \mathbf{loop}\ \boldsymbol{\beta}(\vec{x})\{\vec{y}.\ p[\boldsymbol{\alpha} \setminus \vec{u}.q]\};$$
$$f(\vec{x})\{\vec{y}_i.\ p_i\}_i[\boldsymbol{\alpha} \setminus \vec{u}.q] \equiv f(\vec{x})\{\vec{y}_i.\ p_i[\boldsymbol{\alpha} \setminus \vec{u}.q]\}_i.$$

Proposition 10 (Substitution rules). *The following are derived rules.*

$$\textit{VARIABLE SUBSTITUTION}\quad \frac{\Gamma_1, (\vec{x} : \vec{X}), \Gamma_2 \vdash p : \Delta \qquad (\vec{u} : \vec{X}) \in \Gamma}{\Gamma_1, \Gamma, \Gamma_2 \vdash p[\vec{x} \setminus \vec{u}] : \Delta} \qquad \textit{LABEL SUBSTITUTION}\quad \frac{\Gamma \vdash p : (\boldsymbol{\alpha} : \vec{X}), \Delta \qquad (\vec{u} : \vec{X}), \Gamma \vdash q : \Delta', \Delta}{\Gamma \vdash p[\boldsymbol{\alpha} \setminus \vec{u}.q] : \Delta', \Delta}$$

2.4 Interchange and Loop axioms

The *interchange* axiom declares that applying a term p and then a term q on each of its branches—and independently of the branch—is the same as applying the term q and then the term p on each of its branches, as long as the variables that both generators use and create are separate.

Axiom 11 (Interchange). Terms of the language must satisfy the following axiom, where the first term have indices $\Delta_1 = (\boldsymbol{\alpha}_1 : \vec{U}_1), ..., (\boldsymbol{\alpha}_n : \vec{U}_n)$ and $\Delta_2 = (\boldsymbol{\beta}_1 : \vec{V}_1), ..., (\boldsymbol{\beta}_m : \vec{V}_m)$, and the resulting equation uses the tensor of both indices, i.e. $\Delta_1 \otimes \Delta_2 = (\boldsymbol{\gamma}_{1,1} : \vec{U}_1, \vec{V}_1), ..., (\boldsymbol{\gamma}_{n,m} : \vec{U}_n, \vec{V}_m)$.

$$\textsc{Interchange}\quad \frac{\Gamma_1 \vdash p : \Delta_1 \qquad \Gamma_2 \vdash q : \Delta_2}{\Gamma_1, \Gamma_2 \vdash p[\boldsymbol{\alpha}_i \setminus \vec{u}_i.q[\boldsymbol{\beta}_j \setminus \vec{v}_j.\boldsymbol{\gamma}_{i,j}(u_i, v_j)]]_i \ \equiv\ q[\boldsymbol{\beta}_j \setminus \vec{v}_j.p[\boldsymbol{\alpha}_i \setminus \vec{u}_i.\boldsymbol{\gamma}_{i,j}(u_i, v_j)]]_j : \Delta_1 \otimes \Delta_2}$$

Remark 12 (Premonoidal and monoidal categories). The interchange axiom distinguishes two possible semantic universes: premonoidal categories and *monoidal categories*. In this text, we will be mostly concerned with *monoidal categories* (those for which the interchange axiom holds), but dropping the interchange axiom does recover a language for the premonoidal case.

The following three axioms (Theorem 13) all concern the behaviour of loops. They are inspired by the axioms of *Conway theories* ([Has97, SP00], which are *traced cartesian multicategories*), only adapted to the distributive setting.

Axiom 13 (Loop axioms)**.** Terms of the language must satisfy the following three axioms.

$$\textsc{Dinaturality}\quad \frac{(\vec{x} : \vec{X}) \in \Gamma \qquad (\vec{u} : \vec{X}), \Gamma \vdash p : (\boldsymbol{\beta} : \vec{Y}), \Delta \qquad (\vec{v} : \vec{Y}), \Gamma \vdash q : (\boldsymbol{\alpha} : \vec{X}), \Delta}{\Gamma \vdash \textbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p[\boldsymbol{\beta} \setminus \vec{v}.q]\} \equiv p[\boldsymbol{\beta} \setminus \vec{y}.\textbf{loop}\ \beta(\vec{y})\{\vec{v}.\ q[\boldsymbol{\alpha} \setminus \vec{u}.p]\}]}$$

$$\textsc{Diagonal}\quad \frac{(\vec{x} : \vec{X}) \in \Gamma \qquad (\vec{u} : \vec{X}), \Gamma \vdash p : (\boldsymbol{\beta} : \vec{X}), (\boldsymbol{\alpha} : \vec{X}), \Delta}{\Gamma \vdash \textbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.\ \textbf{loop}\ \boldsymbol{\beta}(\vec{u})\{\vec{u}.\ p\}\} \equiv \textbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.\ p[\boldsymbol{\beta} \setminus \vec{v}.\boldsymbol{\alpha}(\vec{v})]\} : \Delta}$$

$$\textsc{Uniformity}\quad \frac{\begin{array}{c}(\vec{u} : \vec{X}), \Gamma \vdash \ell : (\boldsymbol{\beta}_1 : \vec{Y}_1), \ldots, (\boldsymbol{\beta}_m : \vec{Y}_m) \qquad (\vec{u} : \vec{X}), \Gamma \vdash p : (\boldsymbol{\gamma} : \vec{X}), \Delta \\ (\vec{v}_i : \vec{Y}_i), (\vec{x} : \vec{X}), \Gamma \vdash q_i : (\boldsymbol{\delta}_i : \vec{Y}_i), \Delta \\ (\vec{x} : \vec{X}) \in \Gamma \qquad (\vec{u} : \vec{X}), \Gamma \vdash p[\boldsymbol{\gamma} \setminus \vec{u}.\ell] \equiv \ell[\boldsymbol{\beta}_i \setminus \vec{v}_i.q_i]_i : (\boldsymbol{\beta}_1 : \vec{Y}_1), \ldots, (\boldsymbol{\beta}_m : \vec{Y}_m), \Delta\end{array}}{\Gamma \vdash \textbf{loop}\ \boldsymbol{\gamma}(\vec{x})\{\vec{u}.p\} \equiv \ell[\vec{u} \setminus \vec{x}][\boldsymbol{\beta}_i \setminus \textbf{loop}\ \boldsymbol{\delta}_i(\vec{y}_i)\{\vec{v}_i.q_i\}] : \Delta}$$

The main consequence of the previous loop axioms is that loops are fixed points.

Proposition 14 (Fixpoint rule)**.** *Looping on a label,* $\textbf{loop}\ \alpha(\vec{x})\{\vec{u}.p\}$*, is a fixed-point for substitution on that label,* $p[\boldsymbol{\alpha} \setminus \bullet]$*, for any term* p*. In other words, the following is a derived rule.*

$$\textit{Fixpoint}\quad \frac{(\vec{x} : \vec{X}) \in \Gamma \qquad (\vec{u} : \vec{X}), \Gamma \vdash p : (\boldsymbol{\alpha} : \vec{X}), \Delta}{\Gamma \vdash \textbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\} \equiv p[\vec{u} \setminus \vec{x}][\boldsymbol{\alpha} \setminus \textbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\}] : \Delta}$$

2.5 Derived structural rules

We do not need to impose the usual structural rules: these are consequences of how our terms were constructed to start with. This has the advantage of simplifying some proofs later, where will not have to separately check that our constructions preserve structural rules.

Proposition 15 (Label exchange, contraction, and weakening)**.** *Exchange, contraction, and weakening for labels are derivable.*

$$\textit{lblExchange}\quad \frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha}_1 : \Psi_1), (\boldsymbol{\alpha}_2 : \Psi_2), \Delta_2}{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha}_2 : \Psi_2), (\boldsymbol{\alpha}_1 : \Psi_1), \Delta_2}$$

$$\textit{lblContraction}\quad \frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha}_1 : \Psi), (\boldsymbol{\alpha}_2 : \Psi), \Delta_2}{\Gamma \vdash \mathsf{lCntr}_{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi), \Delta_2}$$

$$\textit{lblWeakening}\quad \frac{\Gamma \vdash p : \Delta_1, \Delta_2}{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha} : \Psi), \Delta_2}$$

Proposition 16 (Index tensor exchange, contraction, weakening)**.** *Exchange, copying, and discarding for variables on the index are derivable.*

$$\textit{rExchange}\quad \frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X_1, X_2, \Psi_2), \Delta_2}{\Gamma \vdash \mathsf{rExch}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X_2, X_1, \Psi_2), \Delta_2}$$

$$\textit{rCopying}\quad \frac{\Gamma \vdash p : \Delta_1, (: \Psi_1, X, \Psi_2), \Delta_2}{\Gamma \vdash \mathsf{rCopy}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X, X, \Psi_2), \Delta_2}$$

$$\textit{rDiscarding}\quad \frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X, \Psi_2), \Delta_2}{\Gamma \vdash \mathsf{rDisc}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, \Psi_2), \Delta_2}$$

Proposition 17 (Variable exchange and contraction). *Variable exchange, variable contraction, and variable weakening are derivable.*

$$\textsc{varExchange}\quad \frac{\Gamma_1, (x : X), (y : Y), \Gamma_2 \vdash p : \Delta}{\Gamma_1, (y : Y), (x : X), \Gamma_2 \vdash p : \Delta} \qquad \textsc{varContraction}\quad \frac{\Gamma_1, (x_1 : X), (x_2 : X), \Gamma_2 \vdash p : \Delta}{\Gamma_1, (x : X), \Gamma_2 \vdash p[x_1, x_2 \setminus x, x] : \Delta} \qquad \textsc{varWeakening}\quad \frac{\Gamma_1, \Gamma_2 \vdash p : \Delta}{\Gamma_1, (x : X), \Gamma_2 \vdash p : \Delta}$$

2.6 Posetal reasoning

Program logics will require not only that we reason about equality, but also about different notions of implication and dominance that only share the common structure of partially ordered sets preserved by the term constructors. For this, it is also convenient to assume a partially ordered set in the generators of the language. Most of our semantic examples will actually form directed-complete partial orders (*dcpo*'s) but, strictly speaking, we do not need them to do so.

Definition 18 (Posetal distributive signature). A *posetal distributive signature*, $(\mathcal{B}, \mathcal{G}, \leq)$, is a distributive signature whose sets of generators are endowed with a poset structure.

Axiom 19 (Posetal reasoning). The following are the primitive rules for posetal reasoning.

$$\textsc{Return}\quad \frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad (\boldsymbol{\alpha} \colon X_1, \ldots, X_n) \in \Delta}{\Gamma \vdash \boldsymbol{\alpha}(x_1, \ldots, x_n) \ \leq\ \boldsymbol{\alpha}(x_1, \ldots, x_n) : \Delta}$$

$$\textsc{Loop}\quad \frac{\{(\vec{x} : \vec{X}) \in \Gamma\} \qquad \Gamma \vdash p \leq q \ : \boldsymbol{\gamma}(X_1, \ldots, X_n), \Delta}{\Gamma \vdash\ (\mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\}) \ \leq\ (\mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.q\}) : \Delta}$$

$$\textsc{Generator}\ (f)\quad \frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad \{\vec{y}_i : \vec{Y}_i, \Gamma \vdash p_i \leq q_i : \Delta\}_{i=1}^{\ell} \qquad f \leq g}{\Gamma \vdash f(\vec{x})\{\vec{y}_i.p_i\}_{i=1}^{\ell} \ \leq\ g(\vec{x})\{\vec{y}_i.q_i\}_{i=1}^{\ell} : \Delta}$$

We ask for two additional conditions—inspired by our intended semantics—declaring the top and bottom elements of this preorder to be the empty return and the diverging loop, respectively.

$$\textsc{Top}\quad \frac{\Gamma \vdash p : (\boldsymbol{\alpha} : ())}{\Gamma \vdash p \leq\ \boldsymbol{\alpha}() : (\boldsymbol{\alpha} : ())} \qquad \textsc{Bottom}\quad \frac{\Gamma \vdash p : \Delta}{\Gamma \vdash\ \mathbf{loop}\ \boldsymbol{\alpha}()\{\boldsymbol{\alpha}()\} \ \leq\ p : \Delta}$$

The final ingredient is for loops to be considered not only up to uniformity but up to both posetal translations of the uniformity rule. This is captured by the following posetal uniformity axioms.

Axiom 20 (Posetal uniformity). *Posetal uniformity* consists of the following pair of axioms.

$$\textsc{Backward posetal uniformity}\quad \frac{(\vec{u} : \vec{X}), \Gamma \vdash p[\boldsymbol{\gamma} \setminus \vec{u}.\ell] \leq \ell[\boldsymbol{\beta}_i \setminus \vec{v}_i.q_i]_i : (\boldsymbol{\beta}_1 : \vec{Y}_1), \ldots, (\boldsymbol{\beta}_m : \vec{Y}_m), \Delta}{\Gamma \vdash \mathbf{loop}\ \boldsymbol{\gamma}(\vec{x})\{\vec{u}.p\} \leq \ell[\vec{u} \setminus \vec{x}][\boldsymbol{\beta}_i \setminus \mathbf{loop}\ \boldsymbol{\delta}_i(\vec{y}_i)\{\vec{v}_i.q_i\}] : \Delta}$$

$$\textsc{Forward posetal uniformity}\quad \frac{(\vec{u} : \vec{X}), \Gamma \vdash \ell[\boldsymbol{\beta}_i \setminus \vec{v}_i.q_i]_i \leq p[\boldsymbol{\gamma} \setminus \vec{u}.\ell] : (\boldsymbol{\beta}_1 : \vec{Y}_1), \ldots, (\boldsymbol{\beta}_m : \vec{Y}_m), \Delta}{\Gamma \vdash \ell[\vec{u} \setminus \vec{x}][\boldsymbol{\beta}_i \setminus \mathbf{loop}\ \boldsymbol{\delta}_i(\vec{y}_i)\{\vec{v}_i.q_i\}] \leq \mathbf{loop}\ \boldsymbol{\gamma}(\vec{x})\{\vec{u}.p\} : \Delta}$$

3 Guards, predicates and commands

Program triples, $\{p\}\ c\ \{q\}$, contain three elements, but of different nature. To start with, while the middle element, c, is a command modifying a state of the program, both p and q are conditions that do not produce new values. In terms of categories, commands are endomorphisms $c \colon X \to X$ on a fixed type X of program states, while conditions will be—depending on the logic—either predicates, $p, q \colon X \to I$, or states, $p, q \colon I \to X$.

It is tempting to conflate predicates and states. In non-deterministic semantics, for instance, they coincide: a function from X to $\mathcal{P}(1)$ is the same as a function from 1 to $\mathcal{P}(X)$. We must resist this temptation. Already in the stochastic case, a function $p \colon X \to \mathcal{D}(1)$ assigns a number in the unit interval to each element, $p(x) \in [0, 1]$, representing the probability that x satisfies the property p; on the other hand, a function $s \colon 1 \to \mathcal{D}(X)$ is a distribution: it not only assigns an number to each element, but explicitly asks them to add up to 1, as they represent the probability that the different events in X happen.

The second temptation is to conflate predicates with the conditions that commands use in their "if-else" clauses: what we call guards. Guards, however, are morphisms $b \colon X \to 1 + 1$. They do not deal only with choosing whether some condition holds or not, but must decide on which of the branches to follow.

In many models, guards and predicates can be confused. For instance, a partial function $X \to 1$ is the same thing as a total function $X \to 1 + 1$; the first has the form of a predicate, the second that of a guard. However, this is not true in general [Jac18, Proposition 11 and Lemma 14] and it is by carefully distinguishing them that we get a consistent algebra that works across probabilistic, partial, or relational models.

3.1 Guards

Definition 21 (Guard combinators)**.** Guards are terms of the form $\Gamma \vdash b : \Omega$, for an arbitrary context $\Gamma = (x_1 : X_1, ..., x_n : X_n)$ and an index of the form $\Omega = (\boldsymbol{\alpha}_1 : (), \boldsymbol{\alpha}_2 : ())$. We introduce the following *guard combinators.*

$$\textsc{Left}\ \frac{}{\Gamma \vdash \mathsf{L} \colon \Omega} \qquad \textsc{Right}\ \frac{}{\Gamma \vdash \mathsf{R} \colon \Omega} \qquad \textsc{And}\ \frac{\Gamma \vdash b_1 \colon \Omega \qquad \Gamma \vdash b_2 \colon \Omega}{\Gamma \vdash b_1 \wedge b_2 \colon \Omega} \qquad \textsc{Or}\ \frac{\Gamma \vdash b_1 \colon \Omega \qquad \Gamma \vdash b_2 \colon \Omega}{\Gamma \vdash b_1 \vee b_2 \colon \Omega} \qquad \textsc{Not}\ \frac{\Gamma \vdash b \colon \Omega}{\Gamma \vdash (\neg b) \colon \Omega}$$

$$\textsc{Pick}\ \frac{\Gamma \vdash b : \Omega \qquad \Gamma \vdash t_1 : \Delta \qquad \Gamma \vdash t_2 \colon \Delta}{\Gamma \vdash [b]\{t_1\}\{t_2\} \colon \Delta}$$

Proposition 22. *Guard combinators are derived constructs, defined as follows.*

$$[b]\{t_1\}\{t_2\} \equiv b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus t_1, t_2];$$
$$\mathsf{L} \equiv \boldsymbol{\alpha}_1(); \qquad \mathsf{R} \equiv \boldsymbol{\alpha}_2(); \qquad (\neg b) \equiv b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_1];$$
$$(b_1 \wedge b_2) \equiv b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2, b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_2]]; \qquad (b_1 \vee b_2) \equiv b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_1], b_2];$$

Proposition 23. *Guards form a pair of commutative monoids, and negation is an involutive homomorphism between them.*

$$b_1 \wedge b_2 \equiv b_2 \wedge b_1; \qquad (b_1 \wedge b_2) \wedge b_3 \equiv b_1 \wedge (b_2 \wedge b_3); \qquad b \wedge \mathsf{L} \equiv b;$$
$$b_1 \vee b_2 \equiv b_2 \vee b_1; \qquad (b_1 \vee b_2) \vee b_3 \equiv b_1 \vee (b_2 \vee b_3); \qquad b \vee \mathsf{R} \equiv b;$$
$$\neg(b_1 \wedge b_2) \equiv \neg b_2 \vee \neg b_1; \qquad \neg(\neg b) \equiv b.$$

For any total guard, $\Gamma \vdash b_t : \Omega$, we additionally have the annihilator rules, $b_t \wedge \mathsf{R} \equiv \mathsf{R}$ and $b_t \vee \mathsf{L} \equiv \mathsf{L}$. For any deterministic guard, $\Gamma \vdash b_d : \Omega$, we additionally have the idempotency rules. $b_d \wedge b_d \equiv b_d$ and $b_d \vee b_d \equiv b_d$.

3.2 Predicates

Definition 24 (Predicate combinators). Predicates are terms of the form $\Gamma \vdash p : \Upsilon$, for an arbitrary context $\Gamma = (x_1 : X_1, ..., x_n : X_n)$ and an index of the form $\Upsilon = (\boldsymbol{v} : ())$. We introduce the following *predicate combinators*.

$$\textsc{Top}\ \frac{}{\Gamma \vdash \top : \Upsilon} \qquad \textsc{Bot}\ \frac{}{\Gamma \vdash \bot : \Upsilon} \qquad \textsc{And}\ \frac{\Gamma \vdash p : \Upsilon \qquad \Gamma \vdash q : \Upsilon}{\Gamma \vdash p \wedge q : \Upsilon} \qquad \textsc{Conditional}\ \frac{\Gamma \vdash p : \Upsilon \qquad \Gamma \vdash q : \Upsilon}{\Gamma \vdash p +_b q : \Upsilon}$$

$$\textsc{Guard}\ \frac{\Gamma \vdash b : \Omega}{\Gamma \vdash b^{\#} : \Upsilon} \qquad \textsc{Substitution}\ \frac{\Gamma \vdash p : \Upsilon \qquad \Gamma \vdash e : (\varepsilon : X_i) \qquad (x_i : X_i) \in \Upsilon}{\Gamma \vdash p[x_i \setminus e] : \Upsilon}$$

Proposition 25. *Predicate combinators are derived constructs, defined as follows.*

$$\top \equiv \boldsymbol{v}(); \qquad \bot \equiv \mathbf{loop}\ \boldsymbol{\omega}()\{\boldsymbol{\omega}()\}; \qquad (p \wedge q) \equiv p[\boldsymbol{v} \setminus q]; \qquad (p +_b q) \equiv [b]\{p\}\{q\};$$
$$b^{\#} \equiv [b]\{\top\}\{\bot\}; \qquad p[x_i \setminus e] \equiv e[\varepsilon \setminus x_i.p].$$

Proposition 26. *The following equations hold for predicate combinators: predicates form a commutative monoid with conjunction and truth, with falsehood as an absorbing element, that distributes over choices.*

$$p \wedge q \equiv q \wedge p; \qquad p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r; \qquad p \wedge \top \equiv p; \qquad p \wedge \bot \equiv \bot;$$
$$p \wedge (q +_b r) \equiv (p \wedge q) +_b (p \wedge r).$$

For any total predicate, $\Gamma \vdash p_t : \Upsilon$, we have it collapse, $p \equiv \top$. For any deterministic predicate, $\Gamma \vdash p_d : \Upsilon$, we have the idempotency rule, $p_d \wedge p_d \equiv p_d$.

3.3 Commands

Definition 27 (Command combinators). Commands are terms of the form $\Gamma \vdash c : \Psi$, for an arbitrary context $\Gamma = (x_1 : X_1, ..., x_n : X_n)$ and an index of the form $\Psi = (\boldsymbol{\eta} : (X_1, ..., X_n))$. We introduce the following *command combinators*, inspired by Winskel's *IMP language* [Win93].

$$\textsc{Skip}\ \frac{}{\Gamma \vdash \mathsf{skip} : \Psi} \qquad \textsc{Abort}\ \frac{}{\Gamma \vdash \mathsf{abort} : \Psi} \qquad \textsc{While}\ \frac{\Gamma \vdash b : \Omega \qquad \Gamma \vdash c : \Psi}{\Gamma \vdash \mathsf{while}\ b\ \mathsf{do}\ c : \Psi} \qquad \textsc{IfElse}\ \frac{\Gamma \vdash b : \Omega \qquad \Gamma \vdash c_1 : \Psi \qquad \Gamma \vdash c_2 : \Psi}{\Gamma \vdash \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \Psi}$$

$$\textsc{Concatenate}\ \frac{\Gamma \vdash c_1 : \Psi \qquad \Gamma \vdash c_2 : \Psi}{\Gamma \vdash (c_1 ; c_2) : \Psi} \qquad \textsc{Assert}\ \frac{\Gamma \vdash p : \Upsilon}{\Gamma \vdash \mathsf{assert}\ p : \Gamma} \qquad \textsc{Variable Assignment}\ \frac{\{(u_i : A_i) \in \Gamma\}_{i=1}^{n} \qquad \{(v_i : A_i) \in \Gamma\}_{i=1}^{n}}{\Gamma \vdash u_1, ..., u_n \coloneqq v_1, ..., v_m : \Psi}$$

$$\textsc{Generator Assignment}\ \frac{\{(u_i : A_i) \in \Gamma\}_{i=1}^{n} \qquad \{(v_j : B_j) \in \Gamma\}_{j=1}^{m} \qquad f \in \Sigma(A_1, ..., A_n; B_1, ..., B_m)}{\Gamma \vdash u_1, ..., u_n \coloneqq f(v_1, ..., v_m) : \Psi}$$

Proposition 28. *Command combinators are derived constructors, defined as follows.*

$$\mathsf{skip} \equiv \boldsymbol{\eta}(\vec{x}); \qquad (c_1 ; c_2) \equiv c_1[\boldsymbol{\eta} \setminus \vec{x}.c_2]; \qquad \mathsf{assert}\ p \equiv p[\boldsymbol{v} \setminus \boldsymbol{\eta}(\vec{x})] \qquad \mathsf{abort} \equiv \mathsf{assert}\ \bot;$$
$$(\vec{u} \coloneqq \vec{v}) = \boldsymbol{\eta}(\vec{x})[\vec{u} \setminus \vec{v}]; \qquad (\vec{u} \coloneqq f(\vec{v})) = f(\vec{v})\{\vec{u}.\boldsymbol{\eta}(\vec{x})\}; \qquad \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \equiv [b]\{c_1\}\{c_2\};$$
$$\mathsf{while}\ b\ \mathsf{do}\ c \equiv \mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\mathsf{if}\ b\ \mathsf{then}\ c[\boldsymbol{\eta} \setminus \vec{x}.\boldsymbol{\alpha}(\vec{x})]\ \mathsf{else\ skip}\};$$

Proposition 29. *The following equations hold for command combinators. In particular, commands form a monoid, with composition and* skip.

$$(c_1 ; c_2) ; c_3 \equiv c_1 ; (c_2 ; c_3); \qquad (c ; \mathsf{skip}) \equiv c \equiv (\mathsf{skip} ; c); \qquad \mathsf{abort}; c \equiv \mathsf{abort} \equiv c ; \mathsf{abort};$$

$$\text{if } \mathbf{L} \text{ then } c_1 \text{ else } c_2 \equiv c_1; \qquad \text{if } \mathbf{R} \text{ then } c_1 \text{ else } c_2 \equiv c_2; \qquad \text{if}(\neg b) \text{ then } c_1 \text{ else } c_2 \equiv \text{if } b \text{ then } c_2 \text{ else } c_1;$$

$$\text{while } b \text{ do } c \equiv \text{if } b \text{ then}(c \,;\, \text{while } b \text{ do } c) \text{ else skip}; \qquad \text{while } b \text{ do abort} \equiv \text{assert } (\neg b)^{\#};$$

$$\text{if } b \text{ then } c_1 \text{ else } c_2 \,;\, d \equiv \text{if } b \text{ then}(c_1; d) \text{ else}(c_2; d);$$

$$\text{assert } p; \text{assert } q \equiv \text{assert}(p \wedge q); \qquad \text{assert } b^{\#} \equiv \text{if } b \text{ then skip else abort};$$

$$\text{assert } \top \equiv \text{skip}; \qquad \text{assert } \bot \equiv \text{abort}; \qquad \text{assert}(p +_b q) = \text{if } b \text{ then}(\text{assert } p) \text{ else}(\text{assert } q)$$

We define a combinator that does not yield an endomorphism but that will be useful in the proofs that employ uniformity.

Definition 30. For a guard b and two arbitrary terms t_1 and t_2, the *branch combinator* is defined as $(\!|b|\!)\{t_1\}\{t_2\} \equiv b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus t_1, t_2]$. Its typing rule is below.

$$\textsc{Branch}\ \frac{\Gamma \vdash b : \Omega \qquad \Gamma \vdash t_1 : \Delta_1 \qquad \Gamma \vdash t_2 : \Delta_2}{\Gamma \vdash (\!|b|\!)\{c_1\}\{c_2\} : \Delta_1, \Delta_2}$$

3.4 States

Definition 31 (States). *States* are terms of the form $\vdash s : \Psi$, implicitly fixing an arbitrary context $\Gamma = (x_1 : X_1, ..., x_n : X_n)$ and taking an index of the form $\Psi = (\boldsymbol{\eta} : (X_1, ..., X_n))$. We introduce the following *state combinators*.

$$\textsc{abort}\ \frac{}{\vdash \bot : \Psi} \qquad \textsc{observe}\ \frac{\vdash s : \Psi \qquad \Gamma \vdash p : \Upsilon}{\vdash s \downharpoonleft p : \Psi} \qquad \textsc{choice}\ \frac{\vdash s : \Psi \qquad \vdash t : \Psi \qquad \vdash b : \Omega}{\vdash s +_b t : \Psi} \qquad \textsc{sample}\ \frac{\vdash s : \Psi \qquad (x : X) \in \Gamma}{\vdash (x \leftarrow s) : \Psi}$$

$$\textsc{cosubstitution}\ \frac{(x : X) \in \Gamma \qquad (u : X) \in \Gamma}{\vdash s(u \setminus x) : \Psi} \qquad \textsc{mute}\ \frac{\vdash s : \Psi \qquad \vdash s_i : (\alpha_i : X_i) \qquad (x_i : X_i) \in \Gamma}{\vdash \coprod_{x_i} s \cdot s_i : \Psi}$$

Proposition 32. *State combinators are derived rules, defined as follows.*

$$\bot \equiv \mathbf{loop}\, \boldsymbol{\alpha}()\{\boldsymbol{\alpha}()\}; \qquad s \downharpoonleft p \equiv (s; \text{assert } p); \qquad s +_b t \equiv b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus s, t];$$

$$(x_i \leftarrow s) \equiv (x_i := s_i()); \qquad s(u \setminus x) \equiv s[\boldsymbol{\eta} \setminus x := u]; \qquad \coprod_{x_1} s \cdot s_i \equiv s[\boldsymbol{\eta} \setminus x_i := s_i()];$$

4 Categorical semantics

After having finally introduced all the components of program logics, this section provides their categorical semantics.

4.1 Premonoidal copy-discard categories

Premonoidal categories [PT97, PR97, Jef97] provide denotational semantics to process theories where the order of execution matters, as it usually does in impure imperative programming. Our multiplicative fragment semantics is inspired by the theory of *Freyd categories* [PT97, Lev22, HJ06], but instead of allowing a distinguished class of cartesian values, we simply ask for the ability to copy and discard variables: those providing this ability are called copy-discard premonoidal categories (see also [Fü99]).

Definition 33 (Premonoidal category). A (strict) *premonoidal category* is a category, $\mathbb{C}$, endowed with a sesquifunctor $(\otimes) \colon (\mathbb{C}, \mathbb{C}) \to \mathbb{C}$ and an object $I \in \mathbb{C}$, that are associative and unital on objects, satisfying $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ and $A \otimes I = A = I \otimes A$, and separately associative and unital on morphisms, satisfying: *(i)* $(f \otimes \mathrm{id}_B) \otimes \mathrm{id}_C = f \otimes (\mathrm{id}_B \otimes \mathrm{id}_C)$; *(ii)* $(\mathrm{id}_A \otimes g) \otimes \mathrm{id}_C = \mathrm{id}_A \otimes (g \otimes \mathrm{id}_C)$; *(iii)* $\mathrm{id}_A \otimes (\mathrm{id}_B \otimes h) = (\mathrm{id}_A \otimes \mathrm{id}_B) \otimes h$; and *(iv)* $\mathrm{id}_I \otimes f = f = f \otimes \mathrm{id}_I$.

Crucially, a premonoidal category does not necessarily satisfy the following *interchange axiom*. We say that a morphism, $f\colon A \to A'$, is *central* whenever, for any morphism $g\colon B \to B'$, the interchange axiom holds:

$$(f \otimes \mathrm{id}_B) ⨟ (\mathrm{id}_{A'} \otimes g) = (\mathrm{id}_{A'} \otimes g) ⨟ (f \otimes \mathrm{id}_{B'}).$$

A *monoidal category* is a premonoidal category where all morphisms are central.

Definition 34 (Copy-discard premonoidal category)**.** A *copy-discard premonoidal category* is a symmetric premonoidal category where each object, X, has a compatible and central cocommutative comonoid structure: a *copy* morphism $\nu_X\colon X \to X \otimes X$ and a *discard* morphism $\varepsilon_X\colon X \to I$, that are associative, $\nu_X ⨟ (\nu_X \otimes \mathrm{id}_X) = \nu_X ⨟ (\mathrm{id}_X \otimes \nu_X)$, unital, $\nu_X ⨟ (\varepsilon_X \otimes \mathrm{id}_X) = \mathrm{id}_X$, commutative, $\nu_X ⨟ \sigma_{X,X} = \nu_X$, and compatible with tensor and unit, $\nu_{X\otimes Y} = (\nu_X \otimes \nu_Y) ⨟ (\mathrm{id}_X \otimes \sigma_{X,Y} \otimes \mathrm{id}_Y)$ and $\varepsilon_{X\otimes Y} = (\varepsilon_X \otimes \varepsilon_Y)$, and $\nu_I = \mathrm{id}_I$ and $\varepsilon_I = \mathrm{id}_I$. A *copy-discard monoidal category* is a copy-discard premonoidal category where all morphisms are central.

Definition 35 (Deterministic and total morphisms)**.** In a copy-discard category, a morphism $f\colon X \to Y$ is *deterministic* if it preserves copying, $f ⨟ \nu_Y = \nu_X ⨟ (f \otimes f)$; it is *total* if it preserves discarding, $f ⨟ \varepsilon_Y = \varepsilon_X$.

Proposition 36 (Grandis [Gra01, Theorem 4.1], Lack [Lac04, §5.1])**.** *Each copy-discard category, $(\mathbb{C}, \otimes, I)$, is endowed with a (non-natural) family of morphisms for each opposite function between finite sets,*

$$f^\star_X\colon \mathbb{C}(X_1, ..., X_n; X_{f(1)}, ..., X_{f(m)}),\ \textit{for each } f \in \mathbf{FinSet}(m; n);$$

these additionally satisfy (i) $f^\star_X \otimes g^\star_Y = (f+g)^\star_{X\otimes Y}$, (ii) $f^\star_X ⨟ g^\star_{X(f)} = (g ⨟ f)^\star_X$, *and* (iii) $\mathrm{id}^\star_X = \mathrm{id}_X$.

Remark 37 (Values and computations). The language here proposed does not define values separately from statements: it is not possible to substitute values for variables. Instead, it is possible to substitute variables, generators by terms, and labels by terms. Nothing—but minimalism—prevents us from adding this distinction; but let us note that it is not necessary for our development.

Example 38. Copy-discard premonoidal categories provide a less expressive but more general alternative to Moggi's *monadic metalanguage* [Mog91]: the Kleisli category of every strong monad, comonad, or distributive law over a cartesian category forms a copy-discard premonoidal category. Copy-discard monoidal categories have encountered applications in probability theory, at the base of *Markov categories.*

However, they lack both *iteration* and *choice*, which makes them too restrictive for fully-fledged imperative programming. We now add choice in the form of cocartesian products: not via cocartesian monoidal categories (which would introduce further redundancy) but via cocartesian multicategories, which reformulate *clones* and *Lawvere theories.*

4.2 Cocartesian multicategories

Multicategories are well-known algebraic structures for the modelling of sequent logic [Her00, Lam68]; their cartesian version, *cartesian multicategories*, is the multi-sorted version of *clones*. We will employ *cartesian multicategories* with a twist: their intended semantics is not in categories we would think of as cartesian, but on the "opposite to a cocartesian category". To emphasize this, we call them cocartesian multicategories.

The structure of copy-discard premonoidal category we just detailed will still be present, but now as an operation on multimorphisms. Cocartesian multicategories that are, at the same time—and in a compatible way—copy-discard premonoidal categories form predistributive multicategories;

respectively, cocartesian multicategories that are at the same time—and in a compatible way—copy-discard categories form distributive multicategories. While these are less studied in the literature, their representable counterparts distributive categories are well-known; we extract coherence results from this literature [Lap06].

Definition 39 (Multicategory). A *multicategory* (or, equivalently, a *comulticategory*), $\mathbb{M}$, is a collection of objects, $\mathbb{M}_{obj}$, together with a collection of multimorphisms, $\mathbb{M}(X; Y_1, \ldots, Y_n)$, for each object, $X \in \mathbb{M}_{obj}$, and each list of objects, $Y_1, \ldots, Y_n \in \mathbb{M}_{obj}$.

For each object, $X \in \mathbb{M}_{obj}$, there must exist an identity morphism, $\mathrm{id}_X : X \to X$; and for each object, $X \in \mathbb{M}_{obj}$, each n-list of objects, $Y_1, \ldots, Y_n \in \mathbb{M}_{obj}$, and each n lists of objects, $Z_{i,1}, \ldots, Z_{i,m_i} \in \mathbb{M}_{obj}$, there exists a composition operation,

$$(\fatsemi) \colon \mathbb{M}(X; Y_1, \ldots, Y_n) \times \prod_{i=0}^{n} \mathbb{M}(Y_i; Z_{i,1}, \ldots, Z_{i,m_i}) \to \mathbb{M}(X; Z_{1,1}, \ldots, Z_{n,m_n}).$$

Composition and identities must satisfy the *unitality* axiom, stating that $\mathrm{id} \fatsemi f = f = f \fatsemi (\mathrm{id}, \ldots, \mathrm{id})$; and the *associativity* axiom, stating that

$$f \fatsemi (g_1 \fatsemi (h_{1,1}, \ldots, h_{1,m_1}), \ldots, g_n \fatsemi (h_{n,1}, \ldots, h_{n,m_n})) = f \fatsemi (g_1 \fatsemi, \ldots, g_n) \fatsemi (h_{1,1}, \ldots, h_{1,m_1}, \ldots, h_{n,1}, \ldots, h_{n,m_n}).$$

Remark 40. Multicategories can be also axiomatized in terms of a composition operation on a single index, which is sometimes more comfortable. We write the single composition operation as $f \fatsemi_i g = f \fatsemi (\mathrm{id}, \ldots, g^{(i)}, \ldots, \mathrm{id})$. It must satisfy *(i)* that $(f \fatsemi_i g) \fatsemi_j h = f \fatsemi_i (g \fatsemi_{j-i+1} h)$ whenever $i \le j \le i + m - 1$ where g has m outputs, and that *(ii)* that $(f \fatsemi_i g) \fatsemi_j h = (f \fatsemi_{j-i+1} h) \fatsemi_i g$ whenever $i + m - 1 < j$.

Lemma 41 (Terms form a multicategory). *Terms, with composition, form a multicategory. The composition of two terms with appropriately matching types, $\Gamma \vdash p \colon \Delta_1, (\boldsymbol{\omega} \colon Y_1, \ldots, Y_m), \Delta_2$ and $(y_1 : Y_1), \ldots, (y_m : Y_m) \vdash q \colon \Delta$, along the label $\boldsymbol{\omega}$, yields a term, $\Gamma \vdash (p \fatsemi_\omega q) \colon \Delta_1, \Delta, \Delta_2$, inductively defined as follows.*

$$\boldsymbol{\omega}(\vec{x}) \fatsemi_\omega q \equiv q[\vec{y} \setminus \vec{x}];$$
$$\boldsymbol{\alpha}(\vec{x}) \fatsemi_\omega q \equiv \boldsymbol{\alpha}(\vec{x}), \quad \textit{for } \boldsymbol{\alpha} \neq \boldsymbol{\omega};$$
$$(\mathtt{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\}) \fatsemi_\omega q \equiv \mathtt{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.(p \fatsemi_\omega q)\};$$
$$(f(\vec{x})\{\vec{y}_i.p_i\}) \fatsemi_\omega q \equiv f(\vec{x})\{\vec{y}_i.(p_i \fatsemi_\omega q)\}.$$

The identity term, $\vec{x} : \vec{X} \vdash \mathrm{id} \colon (\boldsymbol{\alpha} : \vec{X})$, is defined by $\mathrm{id} = \boldsymbol{\alpha}(\vec{x})$.

Proposition 42 (Cut-elimination). *The following* CUT *is a derived rule.*

$$\textsc{Cut}\ \frac{\Gamma \vdash p \colon \Delta_1, (\boldsymbol{\omega} \colon Y_1, \ldots, Y_m), \Delta_2 \qquad (y_1 : Y_1), \ldots, (y_m : Y_m) \vdash q \colon \Delta}{\Gamma \vdash (p \fatsemi_\omega q) \colon \Delta_1, \Delta, \Delta_2}$$

By only considering labels – and forgetting about the variable structure – terms follow the structure of a cocartesian multicategory. This is the equivalent opposite of a *cartesian multicategory* (a *clone*, or a *colored Lawvere theory*). In particular, a cocartesian multicategory is a *symmetric multicategory.*

Definition 43 (Cocartesian multicategory). A *cocartesian multicategory* is a multicategory $\mathbb{M}$ with, for each finite function, $\sigma \colon m \to n$, an action, $(\bullet) \cdot \sigma^* \colon \mathbb{M}(X; Y_{\sigma(1)}, \ldots, Y_{\sigma(m)}) \to \mathbb{M}(X; Y_1, \ldots, Y_n)$, satisfying axioms,

(1) $f \cdot \mathrm{id}^* = f$, and $f \cdot \sigma^* \cdot \tau^* = f \cdot (\sigma ⨟ \tau)^*$;
(2) $g ⨟ (f_1 \cdot \sigma_1^*, ..., f_n \cdot \sigma_n^*) = (g ⨟ (f_1, ..., f_n)) \cdot (\sigma_1 + ... + \sigma_n)^*$;
(3) $g \cdot \sigma^* ⨟ (f_1, ..., f_n) = (g ⨟ (f_{\sigma(1)}, ..., f_{\sigma(m)})) \cdot (\sigma(k_1, ..., k_m))^*$.

Here, by $\sigma(k_1, ..., k_n) \colon k_{\sigma(1)} + ... + k_{\sigma(m)} \to k_1 + ... + k_n$, we denote the block function that acts as the identity on each one of the blocks, and as $\sigma \colon m \to n$ among them [Shu16]. By $\sigma_1 + ... + \sigma_n \colon k_1 + ... + k_n \to k_1' + ... + k_n'$ we denote the coproduct of finite functions. Later, we will use $[\sigma_1, ..., \sigma_n] \colon k_1 + ... + k_n \to k$ to denote the cotupling of functions sharing a codomain.

Proposition 44 (Terms form a cocartesian multicategory). *Terms form a cocartesian multicategory with label substitution. The following rule is derivable and satisfies the axioms in Theorem 43.*

Label coaction
$$\frac{\Gamma \vdash p : (\boldsymbol{\alpha}_1 : \Psi_{\sigma(1)}), ..., (\boldsymbol{\alpha}_m : \Psi_{\sigma(m)})}{\Gamma \vdash p[\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_m \setminus \boldsymbol{\beta}_{\sigma(1)}, ..., \boldsymbol{\beta}_{\sigma(m)}] : (\boldsymbol{\beta}_1 : \Psi_1), ..., (\boldsymbol{\beta}_n : \Psi_n)}$$

4.3 Distributive copy-discard multicategories

Definition 45 (Predistributive multicategory). A (strict) *predistributive multicategory* is a cocartesian multicategory, $(\mathbb{M}, *)$, with a monoid on objects, $(\mathbb{M}_{obj}, \otimes, 1)$, and, additionally, operations

$$(\bullet \rtimes U) \colon \mathbb{M}(X; Y_1, ..., Y_n) \to \mathbb{M}(X \otimes U; Y_1 \otimes U, ..., Y_n \otimes U),$$
$$(U \ltimes \bullet) \colon \mathbb{M}(X; Y_1, ..., Y_n) \to \mathbb{M}(U \otimes X; U \otimes Y_1, ..., U \otimes Y_n),$$

that must satisfy *(i)* left unitality, $(I \ltimes f) = f$, *(ii)* left associativity, $U \ltimes (V \ltimes f) = (U \otimes V) \ltimes f$, *(iii)* right unitality, $(f \rtimes I) = f$, *(iv)* right associativity, $f \rtimes (U \rtimes V) = (f \rtimes U) \rtimes V$, and *(v)* compatibility, $(U \ltimes f) \rtimes V = U \ltimes (f \rtimes V)$.

Definition 46 (Predistributive copy-discard multicategory). A *predistributive copy-discard multicategory* is a predistributive multicategory moreover endowed with the structure of a premonoidal copy-discard category on its unary morphisms.

Lemma 47 (Terms form a predistributive copy-discard multicategory). *Terms form a predistributive copy-discard multicategory. Variable multiwhiskering (multiWhisk-r and multiWhisk-l), where we add the same type to the premises and to each one of the conclusions, are derivable.*

multiWhisk-l
$$\frac{\Gamma \vdash p : (\boldsymbol{\alpha}_1 : \Psi_1), ..., (\boldsymbol{\alpha}_n : \Psi_n)}{\Gamma, (w : X) \vdash X \ltimes p : (\boldsymbol{\alpha}_1 : X, \Psi_1), ..., (\boldsymbol{\alpha}_n : X, \Psi_n)}$$

multiWhisk-r
$$\frac{\Gamma \vdash p : (\boldsymbol{\alpha}_1 : \Psi_1), ..., (\boldsymbol{\alpha}_n : \Psi_n)}{\Gamma, (w : X) \vdash p \rtimes X : (\boldsymbol{\alpha}_1 : \Psi_1, X), ..., (\boldsymbol{\alpha}_n : \Psi_n, X)}$$

The copy-discard category structure follows from the rest of the structural rules (Theorem 16).

Predistributive multicategories, in particular, can compose two morphisms $f \in \mathbb{M}(X; Y_1, ..., Y_n)$ and $f' \in \mathbb{M}(X'; Y_1', ..., Y_m')$ in two different ways: either as $(f \otimes X') ⨟ (X \otimes f', ..., X \otimes f')$, or as $(X \otimes f') ⨟ (f \otimes X, ..., f \otimes X)$. These two cannot coincide; their types do not even match. However, they coincide up to a symmetry: this constitutes the *interchange axiom.*

Definition 48 (Distributive multicategory). A (strict) *distributive multicategory* is a cocartesian multicategory, $(\mathbb{M}, *)$, with a monoid on objects, $(\mathbb{M}_{obj}, \otimes, 1)$, and a tensor operation, $(\otimes)$, taking an n-multimorphism and an m-multimorphism, and yielding an $(n \cdot m)$-multimorphism,

$$\mathbb{M}(X; Y_1, ..., Y_n) \times \mathbb{M}(X'; Y_1', ..., Y_m') \to \mathbb{M}(X \otimes X'; Y_1 \otimes Y_1', ..., Y_1 \otimes Y_m', ..., Y_n \otimes Y_1', ..., Y_n \otimes Y_m'),$$

that satisfies the following axioms: *(i)* associativity, $f \otimes (g \otimes h) = (f \otimes g) \otimes h$, *(ii)* unitality, $f \otimes \mathrm{id} = f = \mathrm{id} \otimes f$, *(iii)* interchange,

$$(f ⨟ (g_1, ..., g_n)) \otimes (f' ⨟ (g_1', ..., g_m')) = (f \otimes f') ⨟ (g_1 \otimes g_1', ..., g_1 \otimes g_m', ..., g_n \otimes g_1', ..., g_n \otimes g_m').$$

Remark 49. In this definition, we choose to order pairs lexicographically—so that $Y_1 \otimes Y'_m$ appears before $Y_n \otimes Y'_m$—but we could have chosen to order pairs *antilexicographically*. This convention corresponds to choosing *left-sesquistrict* over *right-sesquistrict* distributive categories [Lap06].

4.4 Traced distributive multicategories

Definition 50 (Traced distributive multicategory)**.** A *traced distributive multicategory* is a distributive multicategory endowed with a *fixpoint operator*, $\mathsf{fix}\colon \mathbb{M}(X; X, Y_1, ..., Y_n) \to \mathbb{M}(X; Y_1, ..., Y_n)$, satisfying the following axioms:

- *morphism naturality*, $\mathsf{fix}(f) \mathbin{⨟} (a_1, ..., a_n) = \mathsf{fix}(f \mathbin{⨟} (a_1, ..., a_n))$;
- *action naturality*, $\mathsf{fix}(f) \cdot \sigma^* = \mathsf{fix}(f \cdot \mathrm{id}_1 + \sigma^*)$;
- *strength*, $\mathsf{fix}(f \rtimes X) = \mathsf{fix}(f) \rtimes X$ and $\mathsf{fix}(X \ltimes f) = X \ltimes \mathsf{fix}(f)$;
- *duplication*, $\mathsf{fix}(\mathsf{fix}(f)) = \mathsf{fix}(f \cdot [\mathrm{id}_1, \mathrm{id}_1] + \mathrm{id}_n^*)$;
- *dinaturality*, $\mathsf{fix}(f \mathbin{⨟_1} g \cdot [\mathrm{id}_n, \mathrm{id}_n]^*) = g \mathbin{⨟_1} \mathsf{fix}(f \mathbin{⨟_1} g \cdot [\mathrm{id}_n, \mathrm{id}_n, \mathrm{id}_n]^*)$.

Respectively, a *traced distributive copy-discard multicategory* is a traced distributive multicategory endowed with the structure of a copy-discard category on its unary morphisms.

Remark 51 (Terms form a traced multicategory). As expected, terms form a *traced distributive copy-discard multicategory* with looping. We additionally imposed on them the following *uniformity axiom*: the last ingredient to an imperative multicategory.

Definition 52 (Uniform trace)**.** A *uniformly traced distributive multicategory* (or, *Elgot multicategory*), is a traced distributive multicategory additionally satisfying the following *uniformity* axiom: for any appropriately typed multimorphisms, the equality

$$h \mathbin{⨟} (f_1, ..., f_n) \cdot (v_n + \mathrm{id}_m)^* = g \mathbin{⨟} (\mathrm{id}, \overset{n}{\ldots}, \mathrm{id}, h, \overset{m}{\ldots}, h) \cdot (\mathrm{id}_n + v_m)^*;$$

implies the following equality of traces, $h \mathbin{⨟} (\mathsf{fix}(f_1), ..., \mathsf{fix}(f_n)) \cdot v_n^* = \mathsf{fix}(g \cdot v_m^*)$, where we write v_k for the k-cotupling of the identity.

4.5 Imperative multicategories

We can finally introduce the definition of imperative multicategory and immediately employ it to realize the denotational sound and complete semantics of its internal language.

Definition 53 (Imperative multicategory)**.** An *imperative multicategory* is a uniformly traced distributive multicategory, endowed with copy-discard category structure on its unary morphisms.

Theorem 54 (Denotational semantics)**.** *Consider an assignment from a distributive signature $(\mathcal{B}, \mathcal{G})$ to the underlying distributive signature of an imperative multicategory, $(\mathbb{C}_{obj}, \mathbb{C})$, given by an assignment on objects, $(\!|\bullet|\!)_{obj}\colon \mathcal{B} \to \mathbb{C}_{obj}$—which extends to an assignment on lists of types, $[\![\bullet]\!]^{\otimes}\colon \mathsf{List}(\mathcal{B}) \to \mathbb{C}_{obj}$, defined inductively by $[\![]\!]^{\otimes} = I$ and $[\![X, \vec{X}]\!]^{\otimes} = [\![X]\!] \otimes [\![\vec{X}]\!]^{\otimes}$—and an assignment on generators preserving their type,*

$$(\!|\bullet|\!)\colon \mathcal{G}(\vec{X}; \vec{Y}_1, ..., \vec{Y}_n) \to \mathbb{C}((\!|\vec{X}|\!); (\!|\vec{Y}_1|\!) + ... + (\!|\vec{Y}_n|\!)).$$

It extends to an assignment, $[\![\bullet]\!]\colon (\vec{x} : \vec{X} \vdash (\boldsymbol{\alpha}_1 : \vec{Y}_1), ..., (\boldsymbol{\alpha}_1 : \vec{Y}_n)) \to \mathbb{C}([\![\vec{X}]\!]^{\otimes}; [\![\vec{Y}_1]\!]^{\otimes} + ... + [\![\vec{Y}_n]\!]^{\otimes})$, from terms to morphisms of the multicategory $\mathbb{C}$.

Remark 55. Regarding the coproduct, we essentially use the translation between clones and cartesian multicategories [Sze86, Cur12]. Regarding the tensor, we are essentially using the translation from *arrow do-notation* to copy-discard categories.

Theorem 56 (Soundness and completeness)**.** *The denotational semantics is sound and complete for imperative multicategories.*

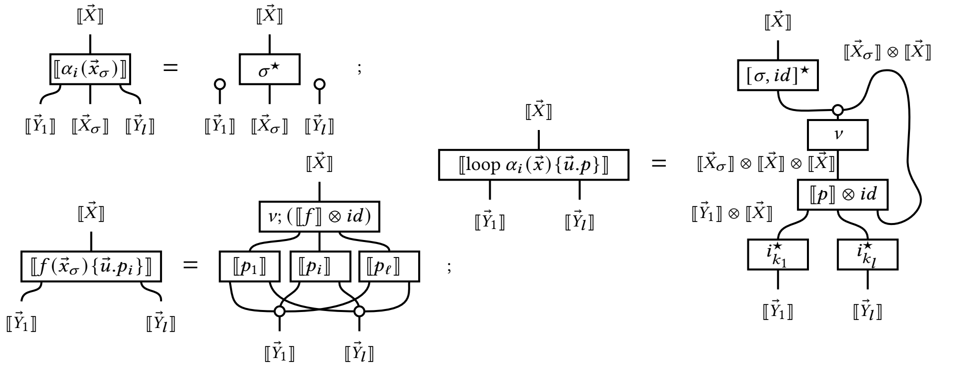


Fig. 1. String diagrams for the semantics of the internal language.

4.6 Posetal imperative multicategories

Reasoning requires an order on morphisms; an order that is respected by all of the operations of the category. We model this by enriching our categories on partially ordered sets.

Definition 57 (Posetal distributive copy-discard multicategory). A *posetal distributive copy-discard multicategory* is a distributive copy-discard multicategory where every set of multimorphisms has a poset structure compatible with composition, tensor, and coproduct actions: for all $f, f' \in \mathbb{M}(X; Y_1, \ldots, Y_n)$ with $f \leq f'$, we have $f \cdot \sigma^* \leq f' \cdot \sigma^*$; for all $g_i, g'_i \in \mathbb{M}(Y_i; Z_{i,1}, \ldots, Z_{i,m_i})$ with $g_i \leq g'_i$, we additionally have $f \fatsemi (g_1, \ldots, g_n) \leq f' \fatsemi (g'_1, \ldots, g'_n)$; for all $h, h' \in \mathbb{M}(X'; Y'_1, \ldots, Y'_n)$ with $h \leq h'$, we additionally have $f \otimes h \leq f \otimes h'$.

Definition 58 (Posetal uniform trace, cf. Hasegawa [Has02]). A *posetal uniform traced distributive multicategory* is a traced distributive multicategory whose underlying multicategory is posetally-enriched and whose fixpoint, additionally, satisfies the *posetal uniformity axiom*: for any appropriately typed multimorphisms, the inequalities

$$h \fatsemi (f_1, \ldots, f_n) \cdot (v_n + \mathrm{id}_m)^* \leq g \fatsemi (\mathrm{id}, \overset{n}{\ldots}, \mathrm{id}, h, \overset{m}{\ldots}, h) \cdot (\mathrm{id}_n + v_m)^*;$$
$$h \fatsemi (f_1, \ldots, f_n) \cdot (v_n + \mathrm{id}_m)^* \geq g \fatsemi (\mathrm{id}, \overset{n}{\ldots}, \mathrm{id}, h, \overset{m}{\ldots}, h) \cdot (\mathrm{id}_n + v_m)^*;$$

imply, respectively, the following inequalities of traces,

$$h \fatsemi (\mathsf{fix}(f_1), \ldots, \mathsf{fix}(f_n)) \cdot v_n^* \leq \mathsf{fix}(g \cdot v_m^*), \quad \text{and} \quad h \fatsemi (\mathsf{fix}(f_1), \ldots, \mathsf{fix}(f_n)) \cdot v_n^* \geq \mathsf{fix}(g \cdot v_m^*).$$

Finally, let us introduce the structure we use for program logics: posetal imperative categories. These express all the constructs of imperative programs but also the logical operations of program logics.

Definition 59 (Posetal imperative multicategory). A *posetal imperative multicategory* is a posetal distributive copy-discard multicategory with posetal uniform trace, and additionally satisfying: *(i)* that its zero map is the least element of any set of multimorphisms, and *(ii)* the discarding map is the top element any set of unary morphisms to the monoidal unit.

4.7 Examples, and representability

Most of our examples have still an extra property: the multicategory is *representable*, meaning that multimorphisms correspond to morphisms to a tensor object (the coproduct). Formally, a multicategory is *representable* when it has, for every list of objects, $Y_1, \ldots, Y_n \in \mathbb{M}_{obj}$, an object $Y_1 + \ldots + Y_n \in \mathbb{M}_{obj}$, and a family of morphisms $\mathrm{case}_n \colon Y_1 + \ldots + Y_n \to Y_1, \ldots, Y_n$ closed under composition and inducing an isomorphism $\mathbb{M}(X; Y_1 + ... + Y_n) \cong \mathbb{M}(X; Y_1, ..., Y_n)$ [Her00, §7]. In a cocartesian multicategory, under this isomorphism, we obtain maps $\mathsf{inj}_{i,n} \colon Y_i \to Y_1 + ... + Y_n$.

We may explicitly impose this property by asking for two families of generators, $\mathsf{case}_n \in \mathcal{G}(Y_1 + ... + Y_n; Y_1, ..., Y_n)$ and $\mathsf{inj}_{i,n} \in \mathcal{G}(Y_i; Y_1 + ... + Y_n)$, which must be total, deterministic, and central, and moreover satisfy the following equations [Her00, Definition 8.1].

- $\mathsf{case}_n(u)\{y_i.\mathsf{inj}_{i,n}(y_i)\{u.\boldsymbol{\alpha}(u)\}\}_{i=0}^n \equiv \boldsymbol{\alpha}(u)$;
- $\mathsf{inj}_{i,n}(x_i)\{u.\mathsf{case}_n(u)\{y_i.\boldsymbol{\alpha}_i(y_i)\}\} \equiv \boldsymbol{\alpha}_i(x_i)$;
- $\mathsf{case}_1(u)\{u.\boldsymbol{\alpha}(u)\} \equiv \boldsymbol{\alpha}(u)$;
- $\mathsf{case}_n(u)\{x_i.\mathsf{case}_m(x_i)\{y_{i,j}.\boldsymbol{\alpha}_{i,j}(y_{i,j})\}\} \equiv \mathsf{case}_{n\cdot m}(u)\{y_{i,j}.\boldsymbol{\alpha}_{i,j}(y_{i,j})\}$;

Definition 60 (Imperative category)**.** An *imperative category* is an imperative multicategory with representable coproducts.

Remark 61. Every multicategory freely induces a representable multicategory; every imperative multicategory freely induces an imperative category. The rest of this section looks at some examples of posetal imperative categories. As common in program semantics, these are Kleisli categories of commutative monads.

Lemma 62. *In a distributive copy-discard category, the structure morphisms of coproducts, μ and ζ, are total and deterministic.*

Definition 63. A *monad* on a category $\mathbb{C}$ is a triple $(T, \eta, (-)^{>})$ of a functor $T \colon \mathbb{C} \to \mathbb{C}$, a family of morphisms $\eta_X \colon X \to T(X)$ indexed by objects X of $\mathbb{C}$, and an operation on hom-sets $(-)^{>} \colon \mathbb{C}(X, TY) \to \mathbb{C}(TX, TY)$ satisfying *(i)* $\eta_X^{>} = \mathrm{id}_{TX}$, *(ii)* $\eta_X ⨾ f^{>} = f$, and *(iii)* $f^{>} ⨾ g^{>} = (f ⨾ g^{>})^{>}$.

The Kleisli category of a monad $T \colon \mathbb{C} \to \mathbb{C}$ commonly serves as semantics for computations in $\mathbb{C}$ with T-effects [Mog91].

Definition 64. For a monad T on a category $\mathbb{C}$, its *Kleisli category*, $\mathsf{kl}(T)$, has the same objects as $\mathbb{C}$ and the morphisms $X \to Y$ are the morphisms $X \to T(Y)$ in $\mathbb{C}$. Identities are given by the monad unit, η_X, and the composition is defined with Kleisli extensions, $f ⨾ g^{>}$.

We introduce the monads whose Kleisli categories will be our running examples. This section shows that they do indeed have the structure of a posetal imperative category.

Example 65. Consider the category Set of sets and functions. The *maybe* monad on Set acts on objects as $\mathcal{L}(X) = X + 1$; its unit is the inclusion $\eta_X \colon X \to X + 1$; and the Kleisli extension of a function $f \colon X \to Y + 1$ is $f^{>}(x) = f(x)$ for $x \in X$, and $f^{>}(*) = *$, where $*$ denotes the element of 1. Morphisms in its Kleisli category, Par, specify partial functions.

Example 66. Consider the *powerset* monad on Set. Its action on objects is $\mathcal{P}(X) = \{E \subseteq X\}$; its unit $\eta_X(x) = \{x\}$ maps each element $x \in X$ to the singleton $\{x\}$; and the Kleisli extension of a function $f \colon X \to \mathcal{P}(Y)$ is $f^{>}(E) = \{f(x) \in Y \mid x \in E\}$. Morphisms in its Kleisli category, Rel, are relations.

Example 67. Consider the *subdistribution* monad on Set. We will consider *countably supported* subdistributions [Jac10, BGL25]. For a set X, these are functions $\sigma \colon X \to [0, 1]$ whose *support*, $\mathrm{supp}(\sigma) = \{x \in X \mid \sigma(x) > 0\}$, is countable and whose total probability mass is at most 1, i.e. $\sum_{x \in X} \sigma(x) \leq 1$. The subdistribution monad maps a set X to the set $\mathcal{D}(X)$ of countably supported subdistributions on X; its unit $\eta_X(x) = \delta_x$ maps each element $x \in X$ to the *Dirac distribution* at point x; and the Kleisli extension of a function $f \colon X \to \mathcal{D}(Y)$ is $f^{>}(\sigma)(y) = \sum_x \sigma(x) \cdot f(x)(y)$. Morphisms in its Kleisli category, Stoch, are discrete stochastic channels.

Example 68. Consider the category $\mathsf{StdBorel}$ of standard Borel spaces and measurable functions between them. A subdistribution on a standard Borel space $(X, \mathcal{A}_X)$ is a measurable function

$\sigma\colon (X, \mathcal{A}_X) \to ([0,1], \mathcal{B})$ whose total probability mass $\sigma(X)$ is at most 1, where $\mathcal{B}$ is the Borel σ-algebra on the interval $[0,1]$. The *subdistribution* monad on StdBorel [Gir82, Pan99] maps a standard Borel space X to the standard Borel space $\mathcal{G}(X)$ of subdistributions on it with the σ-algebra generated by the set of evaluation maps $\mathrm{ev}_U \colon \mathcal{G}(X) \to [0,1]$ for all the measurable subsets U of X.

When the base category has a monoidal structure, we may ask that the monad interacts well with it to ensure that the monoidal structure lifts to the Kleisli category.

Definition 69. A monad T on a symmetric monoidal category $(C, \oplus, I)$ is *strong* if there is a natural transformation $t_{X,Y} \colon X \oplus T(Y) \to T(X \oplus Y)$, the *left strength*, that is compatible with the monoidal structure and with the monad structure: *(i)* $\lambda_{TX} \fatsemi t_{I,X} = T(\lambda_X)$, *(ii)* $t_{X \otimes Y, Z} \fatsemi T(\alpha_{X,Y,Z}) = \alpha_{X,Y,TZ} \fatsemi (\mathrm{id}_X \otimes t_{Y,Z}) \fatsemi t_{X, Y \otimes Z}$, *(iii)* $(\mathrm{id}_X \otimes \eta_Y) \fatsemi t_{X,Y} = \eta_{X \otimes Y}$, and *(iv)* $(\mathrm{id}_X \otimes \mu_Y) \fatsemi t_{X,Y} = t_{X,TY} \fatsemi T(t_{X,Y}) \fatsemi \mu_{X \otimes Y}$, where α, λ and ρ denote the associator, and left and right unitors, and μ denotes the monad multiplication, $\mu_X = \mathrm{id}_{TX}^{>}$.

A strong monad is *commutative* if the two morphism of type $TX \otimes TY \to T(X \otimes Y)$ obtained by composing strengths and symmetries coincide: $t_{TX,Y} \fatsemi T(t'_{X,Y}) \fatsemi \mu_{X \otimes Y} = t'_{X,TY} \fatsemi T(t_{X,Y}) \fatsemi \mu_{X \otimes Y}$, where $t'_{X,Y} = \sigma \fatsemi t \fatsemi T(\sigma)$ is the right strength obtained by composing the left strength t with the symmetry σ.

All the examples of monads in this section are known to be commutative with respect to the cartesian product in their base categories. Any monad is commutative with respect to coproducts. Thus, all their Kleisli categories are distributive copy-discard categories, as the next proposition shows.

Proposition 70. *The Kleisli category of a strong monad $T \colon \mathbb{C} \to \mathbb{C}$ on a distributive copy-discard category $\mathbb{C}$ is also a distributive premonoidal copy-discard category. If the monad T is commutative, then its Kleisli category is a distributive copy-discard category.*

Posetal imperative categories also require a trace for the coproducts. We apply a result that constructs such trace for monads satisfying a condition called *partial additivity* [Jac10]. The conditions for partial additivity are rather technical and we recall them below.

Definition 71 ([Jac10, Definition 4.2])**.** A monad T on a category $\mathbb{C}$ with countable coproducts and products is *partially additive* if its Kleisli category is poset-enriched with a zero object and the morphisms $\beta_{\underline{X}} \colon T(\coprod_n X_n) \to \prod_n T(X_n)$, defined by pairing the canonical maps $\coprod_n X_n \to T(X_i)$, are monic and form a cartesian natural transformation.

Proposition 72 ([Jac10, Example 4.4] and [Jac16, Section 7])**.** *The maybe monad, powerset monad, and subdistributions monad on the distributive category of sets and functions,* Set*, are partially additive. The subdistributions monad on the distributive category* StdBorel *is a partially additive monad.*

While the law of uniformity is well known since at least Hasegawa's work [Has02], the one of posetal uniformity received far less attention (to the best of our knowledge only [BDD25]). We illustrate a result that allows to prove posetal uniformity for a large variety of example, in particular, all those considered in this text. Recall that a $\mathbf{Dcpo}_{\perp}$-enriched category is a category where each homset has countable directed joins and a bottom element that are both preserved by composition.

Our starting point is the following result that ensures the existence of a uniform coproduct trace [Jac10].

Theorem 73. *[Jac10, Theorem 5.2] Let $\mathbb{C}$ be a category with countable coproducts and a monad, $T \colon \mathbb{C} \to \mathbb{C}$, such that*

- *it is a partially additive monad;*
- *its Kleisli category,* $\mathsf{kl}(T)$*, is* $\mathbf{Dcpo}_\perp$*-enriched;*
- *and its Kleisli category,* $\mathsf{kl}(T)$*, has monotone cotuplings;*

then, this Kleisli category is partially additive and has a uniform trace, $(\mathsf{kl}(T), +, 0, \mathrm{tr})$.

Putting together Theorem 73 and Theorem 70, we obtain that these Kleisli categories have almost all the structure that we need.

Corollary 74. *The Kleisli category of a partially additive monad on a distributive category satisfying the assumptions of Theorem 73 is an imperative category.*

With Theorem 74, we are only left to prove posetal uniformity. Starting from Theorem 73, and exploiting a result by Hasuo [Has06] that generalises forward and backward simulations as lax and oplax coalgebra morphisms, we can prove that the monoidal trace of the theorem above is not just a uniform trace but, crucially for our developement, a posetal uniform trace.

Proposition 75. *Under the conditions of Theorem 73, the Kleisli category of a monad,* $\mathsf{kl}(T)$*, has a posetal uniform trace.*

Corollary 76. *The Kleisli categories of the maybe monad, powerset monad, and subdistributions monad on the distributive category* Set*, and of the subdistributions monad on the distributive category* StdBorel *are posetal imperative categories.*

5 Distributive program logics

Program triples are tuples containing a precondition predicate, a command and a postcondition predicate. Program logics are concerned with proving the *validity* of a triple, but what *validity* means depends on the program logic in question and the properties it is concerned with.

For instance, the program triples $\{p\}\ c\ \{q\}$ and $\{s\}\ c\ \{t\}$ may mean any of the inequalities in Figure 2, for a command c, predicates p and q, and states s and t.

	State	Predicate	Assertion
Correctness	$s \mathbin{⨾} c \leq t$	$p \leq c \mathbin{⨾} q$	$\mathsf{assert}\, p \mathbin{⨾} c \leq c \mathbin{⨾} \mathsf{assert}\, q$
Incorrectness	$s \mathbin{⨾} c \geq t$	$p \geq c \mathbin{⨾} q$	$\mathsf{assert}\, p \mathbin{⨾} c \geq c \mathbin{⨾} \mathsf{assert}\, q$

Fig. 2. Inequalities that define validity of program triples $\{p\}\ c\ \{q\}$ or $\{s\}\ c\ \{t\}$.

This section expresses program logics in the language of imperative categories. The next section introduces couplings to cover relational program logics in a similar fashion. This level of generality allows us to instantiate the rules that we prove here in all the examples of Section 4.7.

Each program logic defines validity of triples with one of the inequalities above. Hoare logic [Hoa69] uses $\mathsf{assert}\, p \mathbin{⨾} c \leq c \mathbin{⨾} \mathsf{assert}\, q$, incorrectness logic [dVK11, O'H19] uses $s \mathbin{⨾} c \geq t$, and outcome logic [ZDS23] uses $p \leq c \mathbin{⨾} q$. These are only three of the possibilities outlined above, but nothing prevents us from considering the other ones as well.

The structure of imperative categories allows us to derive rules for any chosen triple shape: the posetal enrichment is crucial for interpreting validity of triples; the categorical structure ensures the SKIP and COMP rules; the monoidal copy-discard structure gives the ASSIGN and SAMPLE rules; the distributive coproducts give the rules for choice; the posetal-uniform trace gives the rules for loops.

5.1 Correctness triples

This section considers assertion-correctness triples. In the category Rel of sets and relations, these are known as *Hoare triples* [Hoa69].

Definition 77 (Assertion-correctness triple). In a posetal imperative category, an *assertion-correctness triple*, $\{p\}\ c\ \{q\}$, consists of a morphism $c \colon X \to Y$, a predicate on the input, $p \colon X \to 1$, and a predicate on the output, $q \colon Y \to 1$, satisfying $\mathsf{assert}\ p \mathbin{⨾} c \leq c \mathbin{⨾} \mathsf{assert}\ q$.

Remark 78. In the imperative category Rel of sets and relations, assertion-correctness triples are equivalent to state-correctness triples: $\mathsf{assert}\ p \mathbin{⨾} c \leq c \mathbin{⨾} \mathsf{assert}\ q$ if and only if $p^{\mathrm{op}} \mathbin{⨾} c \leq q^{\mathrm{op}}$. Predicates have, in general, a richer logic compared to states. Therefore, we choose the former triple shape.

We derive the rules of Hoare logic [Hoa69] as presented by Winskel's reference book [Win93]. Additionally, we include rules for nondeterministic choice and iteration that accommodate examples outside of the category of relations.

Theorem 79. *The following are valid assertion-correctness triples in any posetal imperative category where* $\mathsf{abort} \leq f$ *and* $f \mathbin{⨾} \top \leq \top$ *for all morphisms* f.

$$\textit{SKIP}\ \frac{}{\{p\}\ \mathsf{skip}\ \{p\}} \qquad \textit{COMP}\ \frac{\{p\}\ c_1\ \{q\} \qquad \{q\}\ c_2\ \{r\}}{\{p\}\ c_1 ; c_2\ \{r\}} \qquad \textit{ASSIGN}\ \frac{e \textit{ deterministic and total}}{\{p[u \setminus e]\}\ u \coloneqq e\ \{p\}}$$

$$\textit{CHOICE}\ \frac{\{p\}\ c_1\ \{q\} \qquad \{p\}\ c_2\ \{q\}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}} \qquad \textit{LOOP}\ \frac{\{p\}\ c\ \{p\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{p\}} \qquad \textit{UNROLL}\ \frac{\{p\}\ \mathsf{if}\ b\ \mathsf{then}(c ; \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}\ \{q\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{q\}}$$

$$\textit{IFELSE}\ \frac{\{p \wedge b^{\#}\}\ c_1\ \{q\} \qquad \{p \wedge (\neg b)^{\#}\}\ c_2\ \{q\} \qquad b \textit{ deterministic}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}} \qquad \textit{WHILE}\ \frac{\{b^{\#} \wedge p\}\ c\ \{p\} \qquad b \textit{ deterministic}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{p \wedge (\neg b)^{\#}\}}$$

$$\textit{MONOTONE}\ \frac{p_1 \leq p_2 \qquad \{p_2\}\ c\ \{q_2\} \qquad q_2 \leq q_1}{\{p_1\}\ c\ \{q_1\}} \qquad \textit{AND}\ \frac{\{p_1\}\ c\ \{q_1\} \qquad \{p_2\}\ c\ \{q_2\}}{\{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\}} \qquad \textit{FAIL}\ \frac{}{\{p\}\ \mathsf{abort}\ \{q\}}$$

$$\textit{ASSERT}\ \frac{q \wedge r \leq \bot}{\{p +_b q\}\ \mathsf{assert}\ r\ \{p \wedge b^{\#}\}} \qquad \textit{TOP}\ \frac{}{\{p\}\ c\ \{\top\}} \qquad \textit{BOT}\ \frac{}{\{\bot\}\ c\ \{q\}}$$

5.2 Incorrectness triples

This section considers state-incorrectness triples. In the category Rel of sets and relations, these are known as *reverse Hoare triples* [dVK11] or *incorrectness triples* [O'H19].

Definition 80 (State-incorrectness triple). In a posetal imperative category, a *state-incorrectness triple*, $\{s\}\ c\ \{t\}$, consists of a morphism, $c \colon X \to Y$, a state on the input, $s \colon 1 \to X$, and a state on the output, $t \colon 1 \to Y$, satisfying $s \mathbin{⨾} c \geq t$.

We derive the rules of incorrectness logic [O'H19] in the more general setting of posetal imperative categories. The original incorrectness rules for choices and loops are a particular case of the ones below. They are obtained by setting the guard $b \colon X \to 1 + 1$ to be the relation $\blacktriangleleft = \{(x, 0) \mid x \in X\} \cup \{(x, 1) \mid x \in X\}$, where 0 and 1 denote the two elements of $1 + 1$. Similarly, the nondeterministic assignment rule of incorrectness logic [O'H19] is a particular case of the SAMPLE rule when the state s_0 is chosen to be $\top^{\mathrm{op}}$, the opposite relation of the *true* predicate. The guard $\blacktriangleleft$ and the state $\top^{\mathrm{op}}$ do not exist in general posetal imperative categories, so we present the rules with a generic guard b and a generic state s_0. The rules that we present hold, in particular, for probabilistic examples like Stoch.

We omit the substitution rules in incorrectness logic [O'H19] because they follow from alpha equivalence. We omit the local variable rule because it relies on the existence of the state $\top^{op}$, which does not exist in general. The CONSTANCY rule of incorrectness logic [O'H19] requires the conjunction of preconditions. In copy-discard categories, conjunction of predicates always exists, but not conjunction of states. Thus, we omit this rule. Similarly, the command assume(p) does not necessarily exist in posetal imperative categories. Thus, we substitute the ASSUME rule with the ASSERT rule. The backward variant rule for loops relies on Kleene's theorem for fixpoints. This seems to require more assumptions on the categorical structure, so we decided to omit the rule.

Theorem 81. *The following are valid state-incorrectness triples in any posetal imperative category where* abort $\leq f$ *for all morphisms* f.

$$\textit{SKIP}\ \frac{}{\{s\}\ \mathsf{skip}\ \{s\}} \qquad \textit{COMP}\ \frac{\{s\}\ c_1\ \{t\} \qquad \{t\}\ c_2\ \{r\}}{\{s\}\ c_1 \mathbin{;} c_2\ \{r\}} \qquad \textit{COMP (ERROR)}\ \frac{\{s\}\ c_1\ \{\bot\}}{\{s\}\ c_1 \mathbin{;} c_2\ \{\bot\}}$$

$$\textit{ASSIGN}\ \frac{}{\{s\}\ x \coloneqq y\ \{s(x \setminus y)\}} \qquad \textit{SAMPLE}\ \frac{}{\{s\}\ x \leftarrow s_0\ \{\textstyle\coprod_x s \cdot s_0\}}$$

$$\textit{CHOICE (LEFT)}\ \frac{\{s \downharpoonleft b^{\#}\}\ c_1\ \{t\}}{\{s\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{t\}} \qquad \textit{CHOICE (RIGHT)}\ \frac{\{s \downharpoonleft (\neg b)^{\#}\}\ c_2\ \{t\}}{\{s\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{t\}} \qquad \textit{CONVEX}\ \frac{\{s_1\}\ c\ \{t_1\} \qquad \{s_2\}\ c\ \{t_2\} \qquad b \textit{ constant}}{\{s_1 +_b s_2\}\ c\ \{t_1 +_b t_2\}}$$

$$\textit{ITER ZERO}\ \frac{}{\{s\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{s \downharpoonleft (\neg b)^{\#}\}} \qquad \textit{ITER}\ \frac{\{s \downharpoonleft b^{\#}\}\ c \mathbin{;} \mathsf{while}\ b\ \mathsf{do}\ c\ \{t\}}{\{s\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{t\}}$$

$$\textit{MONOTONE}\ \frac{s_1 \geq s_2 \qquad \{s_2\}\ c\ \{t_2\} \qquad t_2 \geq t_1}{\{s_1\}\ c\ \{t_1\}} \qquad \textit{ASSERT}\ \frac{}{\{s\}\ \mathsf{assert}\ p\ \{s \downharpoonleft p\}} \qquad \textit{FAIL}\ \frac{}{\{s\}\ \mathsf{abort}\ \{\bot\}} \qquad \textit{BOT}\ \frac{}{\{s\}\ c\ \{\bot\}}$$

5.3 Outcome-like triples

This section considers predicate-correctness triples. In Kleisli categories of Set-monads T satisfying some assumptions, these correspond to *outcome triples* [ZDS23].

Definition 82 (Predicate-correctness triples). In a posetal imperative category, a *predicate-correctness triple*, $\{p\}\ c\ \{q\}$, consists of a morphism $c \colon X \to Y$, a predicate on the input, $p \colon X \to 1$, and a predicate on the output, $q \colon Y \to 1$, satisfying $p \leq c \fatsemi q$.

The logic for assertions in outcome logic is richer than the one we consider here: we restrict to the combinators for predicates that come from the categorical structure so that we can interpret the triples and prove their rules in any posetal imperative category. As a consequence, our rules slightly differ from the ones for outcome logic [ZDS23]. As for incorrectness logic, we present the rules with generic guards b as we do not assume the existence of the guard ◀. The CHOICE rule below needs equal postconditions, contrary to that of outcome logic. The structure of posetal imperative categories does not ensure the existence of a predicate $\top^{\oplus}$ that is satisfied by all elements of $T(X)$, including failure. Thus, this structure cannot express the EMPTY and ZERO rules of outcome logic [ZDS23] and implies a different ASSERT rule. We omit the FOR rule as it follows by induction from the rule for compositions and add the SAMPLE rule for nondeterministic assignment.

Theorem 83. *The following are valid predicate-correctness triples in any posetal imperative category where* abort $\le f$ *for all morphisms* f.

SKIP
$$\frac{}{\{p\}\ \mathsf{skip}\ \{p\}}$$

COMP
$$\frac{\{p\}\,c_1\,\{q\} \qquad \{q\}\,c_2\,\{r\}}{\{p\}\,c_1\,;c_2\,\{r\}}$$

ASSIGN
$$\frac{e \text{ deterministic}}{\{p[u \setminus e]\}\,u \coloneqq e\,\{p\}}$$

SAMPLE
$$\frac{}{\{p[u \setminus s]\}\,u \leftarrow s\,\{p\}}$$

UNROLL
$$\frac{\{p\}\ \mathsf{if}\ b\ \mathsf{then}(c\ ;\mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}\ \{q\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{q\}}$$

CHOICE
$$\frac{\{p\}\,c_1\,\{q\} \qquad \{p\}\,c_2\,\{q\} \qquad b \text{ total}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}}$$

IFELSE
$$\frac{\{b^{\#} \wedge p\}\,c_1\,\{q\} \qquad \{(\neg b)^{\#} \wedge p\}\,c_2\,\{q\} \qquad b \text{ total and deterministic}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}}$$

ASSERT
$$\frac{(\neg b)^{\#} \wedge q = \bot \qquad b \text{ deterministic}}{\{p +_b q\}\ \mathsf{assert}\ b^{\#}\ \{p\}}$$

CONVEX
$$\frac{\{p_1\}\,c\,\{q_1\} \qquad \{p_2\}\,c\,\{q_2\} \qquad b \text{ constant}}{\{p_1 +_b p_2\}\,c\,\{q_1 +_b q_2\}}$$

MONOTONE
$$\frac{p_1 \le p_2 \qquad \{p_2\}\,c\,\{q_2\} \qquad q_2 \le q_1}{\{p_1\}\,c\,\{q_1\}}$$

BOT
$$\frac{}{\{\bot\}\,c\,\{q\}}$$

6 Distributive relational program logics

Relational program triples compare pairs of programs in a shared context. They are a tuple of two commands, a precondition on the product of the input types and a postcondition on the product of the output types. As for (not relational) program triples, the validity of relational program triples can be defined in terms of any of the inequalities in Figure 3. This time, p and q are predicates on a product type, s and t are states on a product type, and the commands need to be replaced by couplings of commands as one cannot assume that their effects are independent [BGZB09, BEH+19, ABDG25].

	State	Predicate	Assertion
Relational correctness	$s \fatsemi h^{=} \le t$	$p \le h^{=} \fatsemi q$	$\mathsf{assert}\, p \fatsemi h^{=} \le h^{=} \fatsemi \mathsf{assert}\, q$
Relational incorrectness	$s \fatsemi h^{=} \ge t$	$p \ge h^{=} \fatsemi q$	$\mathsf{assert}\, p \fatsemi h^{=} \ge h^{=} \fatsemi \mathsf{assert}\, q$

Fig. 3. Inequalities that define validity of relational program triples $\{p\}\,c \sim d\,\{q\}$ or $\{s\}\,c \sim d\,\{t\}$, where $h \triangleright c \,\&\, d$ is a coupling of the commands c and d, and $h^{=} = h \fatsemi \pi^{+}_{X \otimes Y}$.

Definition 84. A *coupling* of two morphisms, $f_1 \colon X_1 \to Y_1$ and $f_2 \colon X_2 \to Y_2$ in an imperative category, is a morphism $h \colon X_1 \otimes X_2 \to Y_1 \otimes Y_2 + Y_1 + Y_2$ such that $h \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}] = \pi_1 \fatsemi f_1$ and $h \fatsemi [\pi_2, \mathbb{0}, \mathrm{id}] = \pi_2 \fatsemi f_2$, where $[\pi_1, \mathrm{id}, \mathbb{0}]$ indicates the copairing of the first projection $\pi_1 \colon X_1 \otimes X_2 \to X_1$, the identity id_{X_1} and the zero morphism $\mathbb{0} \colon X_2 \to X_1$. A *strong coupling* is a coupling of the form $h \fatsemi \iota_{Y_1 \otimes Y_2}$, where $\iota_{Y_1 \otimes Y_2} \colon Y_1 \otimes Y_2 \to Y_1 \otimes Y_2 + Y_1 + Y_2$ denotes the coproduct injection.

We write that h is a coupling of f_1 and f_2 as $h \triangleright f_1 \,\&\, f_2$. Given a coupling $h \colon X_1 \otimes X_2 \to Y_1 \otimes Y_2 + Y_1 + Y_2$, define $h^{=} \colon X_1 \otimes X_2 \to Y_1 \otimes Y_2$ by postcomposing with the maps to the zero object, $h^{=} = h \fatsemi \pi^{+}_{Y_1 \otimes Y_2}$.

Remark 85. We spell out the definition of coupling for states in Stoch to show that, in this case, our definition of coupling coincides with the definition of ⋆-coupling for subdistributions [ABDG25]. Two states $s \colon 1 \to X$ and $t \colon 1 \to Y$ in Stoch are two subdistributions $s \in \mathcal{D}(X)$ and $t \in \mathcal{D}(Y)$. A coupling of s and t is a subdistribution $u\colon 1 \to X \times Y + X + Y$ such that $s(x) = \sum_{y \in Y} u(x,y) + u(x,\star)$ and $t(x) = \sum_{x \in X} u(x,y) + u(\star,y)$, where $(x,\star)$ denotes the element x in the second component of the coproduct, and $(\star,y)$ denotes the element y in the third component of the coproduct. A subdistribution on $X \times Y + X + Y$ is the same as a distribution on $X \times Y + X + Y + 1$, thus couplings of states in Stoch coincide with $\star$-couplings of subdistributions [ABDG25]. Similarly, strong couplings coincide with (total) couplings of subdistributions [BGZB09, ABDG25].

Strong couplings enforce the same termination behaviour as total couplings of subdistributions do [ABDG25]. If $h \triangleright f_1 \;\&\; f_2$ is a strong coupling, $(f_1 ⨾ \varepsilon) \otimes \varepsilon = h ⨾ (\varepsilon \otimes \varepsilon) = \varepsilon \otimes (f_2 ⨾ \varepsilon)$, where $f_i ⨾ \varepsilon$ gives the termination predicate of f_i.

Remark 86. When all morphisms are deterministic, then strong couplings trivialise: all strong couplings of f and g need to be $f \otimes g$. This is the case of the category Par of sets and partial functions.

6.1 Relational correctness triples

This section considers relational assertion-correctness triples. In the category Par of sets and partial functions, these correspond to relational Hoare triples [Ben04].

Definition 87 (Relational assertion-correctness triples). In a posetal imperative category, a *relational assertion-correctness triple*, $\{p\}\, c \sim c'\, \{q\}$, consists of two morphisms, $c\colon X \to Y$ and $c'\colon X' \to Y'$, a predicate on the product of the inputs, $p\colon X \otimes X' \to 1$, and a predicate on the product of the outputs, $q\colon Y \otimes Y' \to 1$, such that there exist a coupling, $h \triangleright c \;\&\; c'$, satisfying $\mathsf{assert}\, p ⨾ h^{=} \leq h^{=} ⨾ \mathsf{assert}\, q$.

Benton's work [Ben04] restricts to strong couplings, which simplify in the case of partial functions (Theorem 86). The validity condition of a triple $\{p\}\, c \sim c'\, \{q\}$, thus, simplifies to $\mathsf{assert}\, p ⨾ (c \otimes c') \leq (c \otimes c') ⨾ \mathsf{assert}\, q$. We present the rules in the general case to allow semantics different from partial functions.

Theorem 88. *The following are valid relational assertion-correctness triples in any posetal imperative category where* $\mathsf{abort} \leq f$ *for all morphisms* f*.*

$$\textit{SKIP}\quad \frac{}{\{p\}\ \mathsf{skip} \sim \mathsf{skip}\ \{p\}} \qquad \textit{COMP}\quad \frac{\{p\}\ c_1 \sim d_1\ \{q\} \qquad \{q\}\ c_2 \sim d_2\ \{r\}}{\{p\}\ (c_1 \,;\, c_2) \sim (d_1 \,;\, d_2)\ \{r\}}$$

$$\textit{ASSIGN}\quad \frac{e_1, e_2 \textit{ total and deterministic}}{\{p[(u_1,u_2) \setminus (e_1,e_2)]\}\ (u_1 \coloneqq e_1) \sim (u_2 \coloneqq e_2)\ \{p\}}$$

$$\textit{CHOICE}\quad \frac{\{p\}\ c_1 \sim c_2\ \{q\} \qquad \{p\}\ c_1 \sim d_2\ \{q\} \qquad \{p\}\ d_1 \sim c_2\ \{q\} \qquad \{p\}\ d_1 \sim d_2\ \{q\} \qquad b_1, b_2 \textit{ total}}{\{p\}\ (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\ \{q\}}$$

$$\textit{IFELSE}\quad \frac{\begin{array}{c}\{({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p\}\ c_1 \sim c_2\ \{q\} \\ \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}\ d_1 \sim d_2\ \{q\} \qquad b_1, b_2 \textit{ total and deterministic}\end{array}}{\{(b_1 = b_2) \wedge p\}\ (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\ \{q\}}$$

$$\textit{LOOP}\quad \frac{\{p\}\ c_1 \sim c_2\ \{p\} \qquad \{p\}\ c_1 \sim \mathsf{skip}\ \{p\} \qquad \{p\}\ \mathsf{skip} \sim c_2\ \{p\} \qquad b_1, b_2 \textit{ total}}{\{p\}\ (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\ \{p\}}$$

$$\frac{\{(b_1{}^{\#} \otimes b_2{}^{\#}) \wedge p\}\, c_1 \sim c_2\, \{(b_1 = b_2) \wedge p\} \qquad b_1, b_2 \textit{ total and deterministic}}{\{(b_1 = b_2) \wedge p\}\, (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\, \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}}\ \textsc{while}$$

$$\frac{p_1 \leq p_2 \qquad \{p_2\}\, c \sim d\, \{q_2\} \qquad q_2 \leq q_1}{\{p_1\}\, c \sim d\, \{q_1\}}\ \textsc{monotone} \qquad \frac{\{p\}\, c \sim d\, \{q\}}{\{\sigma \mathbin{;} p\}\, d \sim c\, \{\sigma \mathbin{;} q\}}\ \textsc{symm}$$

$$\frac{e \textit{ total and deterministic}}{\{p[x \setminus e]\}\, (x \coloneqq e) \sim \mathsf{skip}\, \{p\}}\ \textsc{assign-L} \qquad \frac{\{p\}\, c \sim \mathsf{skip}\, \{q\} \qquad \{p\}\, d \sim \mathsf{skip}\, \{q\} \qquad b \textit{ total}}{\{p\}\, (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d) \sim \mathsf{skip}\, \{q\}}\ \textsc{choice-L}$$

$$\frac{\{(b^{\#} \otimes \top) \wedge p\}\, c \sim \mathsf{skip}\, \{q\} \qquad \{((\neg b_1)^{\#} \otimes \top) \wedge p\}\, d \sim \mathsf{skip}\, \{q\} \qquad b \textit{ total and deterministic}}{\{p\}\, (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d) \sim \mathsf{skip}\, \{q\}}\ \textsc{ifelse-L}$$

$$\frac{\{p\}\, c \sim \mathsf{skip}\, \{p\} \qquad b \textit{ total}}{\{p\}\, (\mathsf{while}\ b\ \mathsf{do}\ c) \sim \mathsf{skip}\, \{p\}}\ \textsc{loop-L} \qquad \frac{\{(b^{\#} \otimes \top) \wedge p\}\, c \sim \mathsf{skip}\, \{p\} \qquad b \textit{ total and deterministic}}{\{p\}\, (\mathsf{while}\ b\ \mathsf{do}\ c) \sim \mathsf{skip}\, \{((\neg b)^{\#} \otimes \top) \wedge p\}}\ \textsc{while-L}$$

6.2 Relational incorrectness triples

This section considers relational predicate-incorrectness triples. In the category Stoch of sets and partial stochastic functions, these correspond to quantitative probabilistic relational Hoare triples [ABDG25].

Definition 89 (Relational predicate-incorrectness triples)**.** In a posetal imperative category, a *relational predicate-incorrectness triple*, $\{p\}\ c \sim c'\ \{q\}$, consists of two morphisms, $c\colon X \to Y$ and $c'\colon X' \to Y'$, a predicate on the product of the inputs, $p\colon X \otimes X' \to 1$, and a predicate on the product of the outputs, $q\colon Y \otimes Y' \to 1$, such that there exist a coupling, $h \triangleright c \,\&\, c'$, satisfying $p \geq h^{=} \mathbin{⨟} q$.

We derive the rules of relational predicate-incorrectness logic. Compared to the rules of quantitative probabilistic relational Hoare logic [ABDG25], we do not assume that guards are deterministic, so we derive additional rules for nondeterministic choice and iteration. The STRASSEN rule of quantitative probabilistic relational Hoare logic [ABDG25] is missing as it is a consequence of Strassen's theorem, a characterisation of couplings particular to subdistributions.

For two guards, $b_1\colon X_1 \to 1 + 1$ and $b_2\colon X_2 \to 1 + 1$, we denote with $b_1 = b_2$ the predicate on $X_1 \otimes X_2$ that succeeds when b_1 and b_2 are both true or both false, and fails otherwise. We use $b_1{}^{\#} \otimes b_2{}^{\#}$ to denote the predicate on $X_1 \otimes X_2$ obtained as the monoidal product of $b_1{}^{\#}\colon X_1 \to 1$ and $b_2{}^{\#}\colon X_2 \to 1$. For a predicate $p\colon X_1 \otimes X_2 \to 1$, we indicate with $\sigma \mathbin{;} p\colon X_2 \otimes X_1 \to 1$ the predicate obtained by permuting the inputs.

Theorem 90. *The following are valid relational predicate-incorrectness triples in any posetal imperative category where* abort $\leq f$ *and* $f \mathbin{⨟} \top \leq \top$ *for all morphisms* f*.*

$$\frac{}{\{p\}\ \mathsf{skip} \sim \mathsf{skip}\ \{p\}}\ \textsc{skip} \qquad \frac{\{p\}\, c_1 \sim d_1\, \{q\} \qquad \{q\}\, c_2 \sim d_2\, \{r\}}{\{p\}\, (c_1 \mathbin{;} c_2) \sim (d_1 \sim d_2)\, \{r\}}\ \textsc{comp}$$

$$\frac{}{\{p[(u_1, u_2) \setminus (v_1, v_2)]\}\, (u_1 \coloneqq v_1) \sim (u_2 \coloneqq v_2)\, \{p\}}\ \textsc{assign} \qquad \frac{h \triangleright c_1 \,\&\, c_2}{\{p[(u_1, u_2) \setminus h^{=}]\}\, (u_1 \leftarrow c_1) \sim (u_2 \leftarrow c_2)\, \{p\}}\ \textsc{sample}$$

$$\textsc{Choice}\quad \frac{\{p\}\, c_1 \sim c_2\, \{q\} \qquad \{p\}\, c_1 \sim d_2\, \{q\} \qquad \{p\}\, d_1 \sim c_2\, \{q\} \qquad \{p\}\, d_1 \sim d_2\, \{q\} \qquad b_1, b_2 \textit{ total}}{\{p\}\, (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\, \{q\}}$$

$$\textsc{IfElse}\quad \frac{\begin{array}{c}\{({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p\}\, c_1 \sim c_2\, \{q\} \\ \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}\, d_1 \sim d_2\, \{q\} \qquad b_1, b_2 \textit{ total and deterministic}\end{array}}{\{(b_1 = b_2) \wedge p\}\, (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\, \{q\}}$$

$$\textsc{Loop}\quad \frac{\{p\}\, c_1 \sim c_2\, \{p\} \qquad \{p\}\, c_1 \sim \mathsf{skip}\, \{p\} \qquad \{p\}\, \mathsf{skip} \sim c_2\, \{p\} \qquad b_1, b_2 \textit{ total}}{\{p\}\, (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\, \{p\}}$$

$$\textsc{While}\quad \frac{\{({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p\}\, c_1 \sim c_2\, \{(b_1 = b_2) \wedge p\} \qquad b_1, b_2 \textit{ total and deterministic}}{\{(b_1 = b_2) \wedge p\}\, (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\, \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}}$$

$$\textsc{Monotone}\quad \frac{p_1 \geq p_2 \qquad \{p_2\}\, c \sim d\, \{q_2\} \qquad q_2 \geq q_1}{\{p_1\}\, c \sim d\, \{q_1\}} \qquad \textsc{Choice-L}\quad \frac{\{p\}\, c \sim \mathsf{skip}\, \{q\} \qquad \{p\}\, d \sim \mathsf{skip}\, \{q\} \qquad b \textit{ total}}{\{p\}\, (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d) \sim \mathsf{skip}\, \{q\}}$$

$$\textsc{Symm}\quad \frac{\{p\}\, c \sim d\, \{q\}}{\{\sigma\, ;\, p\}\, d \sim c\, \{\sigma\, ;\, q\}} \qquad \textsc{Assign-L}\quad \frac{}{\{p[x \setminus v]\}\, (x \coloneqq v) \sim \mathsf{skip}\, \{p\}} \qquad \textsc{Sample-L}\quad \frac{c \textit{ total}}{\{p[u \setminus c]\}\, (u \leftarrow c) \sim \mathsf{skip}\, \{p\}}$$

$$\textsc{IfElse-L}\quad \frac{\{(b^{\#} \otimes \top) \wedge p\}\, c \sim \mathsf{skip}\, \{q\} \qquad \{((\neg b_1)^{\#} \otimes \top) \wedge p\}\, d \sim \mathsf{skip}\, \{q\} \qquad b \textit{ total and deterministic}}{\{p\}\, (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d) \sim \mathsf{skip}\, \{q\}}$$

$$\textsc{Loop-L}\quad \frac{\{p\}\, c \sim \mathsf{skip}\, \{p\} \qquad b \textit{ total}}{\{p\}\, (\mathsf{while}\ b\ \mathsf{do}\ c) \sim \mathsf{skip}\, \{p\}} \qquad \textsc{While-L}\quad \frac{\{(b^{\#} \otimes \top) \wedge p\}\, c \sim \mathsf{skip}\, \{p\} \qquad b \textit{ total and deterministic}}{\{p\}\, (\mathsf{while}\ b\ \mathsf{do}\ c) \sim \mathsf{skip}\, \{((\neg b)^{\#} \otimes \top) \wedge p\}}$$

7 Conclusions and future work

We have introduced posetal imperative categories as a principled approach to program logics (Section 4). We have defined a sound and complete syntax for them (Section 2), which allowed us to derive the rules of various existing program logics and relational program logics (Sections 5 and 6).

7.1 Further work

External logic, fibrations, and enrichment. While we focused on the logics given by the internal structure of the category, we could derive more variants if we accept the logic to be external (e.g. the extra operation $\oplus$ of *outcome logic*). In particular, a *fibration* would structure the use of two different categories: one for predicates and one for commands. We considered poset-enriched categories to express program triples. We could extend the treatment to metric-enriched categories to express *quantitative* properties of program behaviour.

Separation logic and premonoidal semantics. The *logic of bunched implications* has semantics in categories that are both cartesian closed and monoidal closed with a second tensor; additional distributivity with coproducts is admissible [OP99]. We believe a careful adaptation of our techniques could derive separation logic from categorical first principles: this could account for its probabilistic versions [BHL19], or be extended to higher-order versions [BTSY06]. The condition that modules have restricted access to some parts of memory [OYR04] may be modelled with premonoidal categories and their internal language [Jef97].

References

[ABDG25] Martin Avanzini, Gilles Barthe, Davide Davoli, and Benjamin Grégoire. A quantitative probabilistic relational Hoare logic. *Proceedings of the ACM on Programming Languages*, 9(POPL):1167–1195, 2025.

[ABH+21] Alejandro Aguirre, Gilles Barthe, Justin Hsu, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. A pre-expectation calculus for probabilistic sensitivity. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–28, 2021.

[AM80] Michael A Arbib and Ernest G Manes. Partially additive categories and flow-diagram semantics. *Journal of Algebra*, 62(1):203–227, 1980.

[AMMO09] Rob Arthan, Ursula Martin, Erik A Mathiesen, and Paulo Oliva. A general framework for sound and complete Floyd-Hoare logics. *ACM Transactions on Computational Logic (TOCL)*, 11(1):1–31, 2009.

[BDD25] Filippo Bonchi, Alessandro Di Giorgio, and Elena Di Lavore. A diagrammatic algebra for program logics. In Parosh Aziz Abdulla and Delia Kesner, editors, *Foundations of Software Science and Computation Structures*, pages 308–330, Cham, 2025. Springer Nature Switzerland.

[BÉ93] Stephen L Bloom and Zoltán Ésik. *Iteration theories*. Springer, 1993.

[BEH+19] Gilles Barthe, Thomas Espitau, Justin Hsu, Tetsuya Sato, and Pierre-Yves Strub. Relational ⋆-liftings for differential privacy. *Logical Methods in Computer Science*, Volume 15, Issue 4, Dec 2019.

[Ben04] Nick Benton. Simple relational correctness proofs for static analyses and program transformations. *ACM SIGPLAN Notices*, 39(1):14–25, 2004.

[BGL25] Nathan Bowler, Sergey Goncharov, and Paul Blain Levy. Probabilistic strategies: Definability and the tensor completeness problem. In *Proceedings of the 40th Annual ACM/IEEE Symposium on Logic in Computer Science (to appear)*, 2025.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 90–101, 2009.

[BHL19] Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–30, 2019.

[BHM00] Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In *International Summer School on Applied Semantics*, pages 42–122. Springer, 2000.

[BK99] Nick Benton and Andrew Kennedy. Monads, effects and transformations. *Electronic Notes in Theoretical Computer Science*, 26:3–20, 1999.

[BKOZB12] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 97–110, 2012.

[BTSY06] Lars Birkedal, Noah Torp-Smith, and Hongseok Yang. Semantics of separation-logic typing and higher-order frame rules for algol-like languages. *Logical Methods in Computer Science*, Volume 2, Issue 5, Nov 2006.

[CH72] Maurice Clint and CAR Hoare. Program proving: Jumps and functions. *Acta informatica*, 1:214–224, 1972.

[CJ13] Dion Coumans and Bart Jacobs. Scalars, monads, and categories. In *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse*. Oxford University Press, 02 2013.

[CLW93] Aurelio Carboni, Stephen Lack, and Robert FC Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84(2):145–158, 1993.

[Coc93] J. Robin B. Cockett. Introduction to distributive categories. *Math. Struct. Comput. Sci.*, 3(3):277–307, 1993.

[Cro12] Roy L. Crole. Alpha equivalence equalities. *Theoretical Computer Science*, 433:1–19, 2012.

[Cur12] Pierre-Louis Curien. Operads, clones, and distributive laws. In *Operads and universal algebra*, pages 25–49. World Scientific, 2012.

[Dij68] Edsger W Dijkstra. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968.

[Dij75] Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[dVK11] Edsko de Vries and Vasileios Koutavas. Reverse hoare logic. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods*, pages 155–171. Springer Berlin Heidelberg, 2011.

[Elg75] Calvin C Elgot. Monadic computation and iterative algebraic theories. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 175–230. Elsevier, 1975.

[Fio93] MP Fiore. A coinduction principle for recursive data types based on bisimulation. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 110–119. IEEE, 1993.

[Fio96] Marcelo P Fiore. A coinduction principle for recursive data types based on bisimulation. *Information and Computation*, 127(2):186–198, 1996.

[Flo93] Robert W Floyd. Assigning meanings to programs. In *Program Verification: Fundamental Issues in Computer Science*, pages 65–81. Springer, 1993.

[Fü99] Carsten Führmann. Direct models of the computational lambda-calculus. In *Proc. MFPS 1999*, 1999.

[GBG25a] Leandro Gomes, Patrick Baillot, and Marco Gaboardi. BiGKAT: an algebraic framework for relational verification of probabilistic programs. In *International Conference on Foundations of Software Science and Computation Structures*, pages 243–264. Springer Nature Switzerland Cham, 2025.

[GBG25b] Leandro Gomes, Patrick Baillot, and Marco Gaboardi. A Kleene algebra with tests for union bound reasoning about probabilistic programs. In *33rd EACSL Annual Conference on Computer Science Logic (CSL 2025)*, pages 35–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

[Gir82] Michèle Giry. A categorical approach to probability theory. In *Categorical aspects of topology and analysis*, pages 68–85. Springer, 1982.

[GK24] Jad Elkhaleq Ghalayini and Neel Krishnaswami. The denotational semantics of ssa. *arXiv preprint arXiv:2411.09347*, 2024.

[Gon10] Sergey Goncharov. *Kleene Monads*. PhD thesis, Universität Bremen, 2010.

[GP99] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. IEEE Computer Society, 1999.

[GP02] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002.

[Gra01] Marco Grandis. Finite sets and symmetric simplicial sets. *Theory and Applications of Categories [electronic only]*, 8:244–252, 2001.

[GRS21] Sergey Goncharov, Christoph Rauch, and Lutz Schröder. A metalanguage for guarded iteration. *Theoretical Computer Science*, 880:111–137, 2021.

[Has97] Masahito Hasegawa. Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. In Philippe de Groote, editor, *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 1997.

[Has02] Masahito Hasegawa. The uniformity principle on traced monoidal categories. In Richard Blute and Peter Selinger, editors, *Category Theory and Computer Science, CTCS 2002, Ottawa, Canada, August 15-17, 2002*, volume 69 of *Electronic Notes in Theoretical Computer Science*, pages 137–155. Elsevier, 2002.

[Has06] Ichiro Hasuo. Generic forward and backward simulations. In *International Conference on Concurrency Theory*, pages 406–420. Springer, 2006.

[Her00] Claudio Hermida. Representable multicategories. *Advances in Mathematics*, 151(2):164–225, 2000.

[HJ06] Chris Heunen and Bart Jacobs. Arrows, like monads, are monoids. In Stephen D. Brookes and Michael W. Mislove, editors, *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2006, Genova, Italy, May 23-27, 2006*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 219–236. Elsevier, 2006.

[HJS06] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace theory. *Electronic Notes in Theoretical Computer Science*, 164(1):47–65, 2006.

[Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Jac10] Bart Jacobs. From coalgebraic to monoidal traces. *Electronic Notes in Theoretical Computer Science*, 264(2):125–140, 2010.

[Jac15] Bart Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. *Logical Methods in Computer Science*, 11, 2015.

[Jac16] Bart Jacobs. Effectuses from monads. *Electronic Notes in Theoretical Computer Science*, 325:169–183, 2016.

[Jac18] Bart Jacobs. From probability monads to commutative effectuses. *Journal of logical and algebraic methods in programming*, 94:200–237, 2018.

[Jef97] Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint at ResearchGate*, 1997.

[Joy95] André Joyal. Free bicompletion of enriched categories. *Mathematical Reports of the Academy of Sciences*, 17(5):213–218, 1995.

[Kam18] Benjamin Lucien Kaminski. *Advanced weakest precondition calculi for probabilistic programs.* PhD thesis, RWTH Aachen University, 2018.

[KK17] Benjamin Lucien Kaminski and Joost-Pieter Katoen. A weakest pre-expectation semantics for mixed-sign expectations. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

[Koz97] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(3):427–443, 1997.

[Lac04] Stephen Lack. Composing props. *Theory and Applications of Categories*, 13(9):147–163, 2004.

[Lam68] Joachim Lambek. Deductive systems and categories i. syntactic calculus and residuated categories. *Math. Syst. Theory*, 2(4):287–318, 1968.

[Lap06] Miguel L Laplaza. Coherence for distributivity. In *Coherence in categories*, pages 29–65. Springer, 2006.

[LCS25] Jack Liell-Cock and Sam Staton. Compositional imprecise probability: A solution from graded monads and markov categories. *Proceedings of the ACM on Programming Languages*, 9(POPL):1596–1626, 2025.

[Lev22] Paul Blain Levy. Call-by-Push-Value. *ACM SIGLOG News*, 9(2):7–29, may 2022.

[LS88] Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*, volume 7. Cambridge University Press, 1988.

[MA12] Ernest G Manes and Michael A Arbib. *Algebraic approaches to program semantics*. Monographs in Computer Science. Springer New York, NY, 2012.

[MCM06] AK McIver, Ernie Cohen, and CC Morgan. Using probabilistic Kleene algebra for protocol verification. In *Relations and Kleene Algebra in Computer Science: 9th International Conference on Relational Methods in Computer Science and 4th International Workshop on Applications of Kleene Algebra, RelMiCS/AKA 2006, Manchester, UK, August 29–September 2, 2006. Proceedings 9*, pages 296–310. Springer, 2006.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[MZ15] Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 3–16, 2015.

[MZ16] Paul-André Melliès and Noam Zeilberger. A bifibrational reconstruction of lawvere's presheaf hyperdoctrine. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 555–564, 2016.

[Nes25] Chad Nester. Elgot categories and abacus programs, 2025.

[O'H19] Peter W O'Hearn. Incorrectness logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–32, 2019.

[Ole83] Frank Joseph Oles. A Category-Theoretic Approach to the Semantics of Programming Languages. *Case Western Reserve University – PhD Thesis*, 1983.

[Olm14] Federico Olmedo. *Approximate Relational Reasoning for Probabilistic Programs*. PhD thesis, Technical University of Madrid, 2014.

[OP99] Peter W O'Hearn and David J Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[ORY01] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Computer Science Logic: 15th International Workshop, CSL 2001 10th Annual Conference of the EACSL Paris, France, September 10–13, 2001, Proceedings 15*, pages 1–19. Springer, 2001.

[OYR04] Peter W O'Hearn, Hongseok Yang, and John C Reynolds. Separation and information hiding. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 268–280, 2004.

[Pan99] Prakash Panangaden. The Category of Markov Kernels. *Electronic Notes in Theoretical Computer Science*, 22:171–187, January 1999.

[PR97] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Math. Struct. Comput. Sci.*, 7(5):453–468, 1997.

[PT97] John Power and Hayo Thielecke. Environments, continuation semantics and indexed categories. In *International Symposium on Theoretical Aspects of Computer Software*, pages 391–414. Springer, 1997.

[RC01] RAG Seely Robin Cockett. Finite sum-product logic. *Theory and Applications of Categories*, 8:63–99, 2001.

[Rey02] John C Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings 17th annual IEEE symposium on logic in computer science*, pages 55–74. IEEE, 2002.

[RKK+23] Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. Probabilistic guarded KAT modulo bisimilarity: Completeness and complexity. In *50th EATCS International Colloquium on Automata, Languages and Programming, ICALP 2023*, page 136. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2023.

[Sat16] Tetsuya Sato. Approximate relational hoare logic for continuous random samplings. *Electronic Notes in Theoretical Computer Science*, 325:277–298, 2016.

[SFH+19] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, 2019.

[Shu16] Michael Shulman. Categorical logic from a categorical point of view, 2016.

[SP00] Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332)*, pages 30–41. IEEE, 2000.

[Sze86] Ágnes Szendrei. Clones in universal algebra. *Les presses de L'universite de Montreal*, 1986.

[Wad98] Philip Wadler. The marriage of effects and monads. In *Proceedings of the third ACM SIGPLAN international conference on Functional programming*, pages 63–74, 1998.

[Wal92] Robert F. C. Walters. An imperative language based on distributive categories. *Math. Struct. Comput. Sci.*, 2(3):249–256, 1992.

[Whi41] Philip M Whitman. Free lattices. *Annals of Mathematics*, 42(1):325–330, 1941.

[Win93] Glynn Winskel. *The formal semantics of programming languages: an introduction.* MIT press, 1993.

[ZDS23] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. Outcome logic: A unifying foundation for correctness and incorrectness reasoning. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1):522–550, 2023.

[ZK22] Linpeng Zhang and Benjamin Lucien Kaminski. Quantitative strongest post: a calculus for reasoning about the flow of quantitative information. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA1):1–29, 2022.

[ZKST25] Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. A demonic outcome logic for randomized nondeterminism. *Proceedings of the ACM on Programming Languages*, 9(POPL):539–568, 2025.

[ZSS24] Noam Zilberstein, Angelina Saliling, and Alexandra Silva. Outcome separation logic: local reasoning for correctness and incorrectness with computational effects. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):276–304, 2024.

A Proofs for Section 2 (An internal distributive language)

Let us restate the rules of the lanugage in a more compact way, using vectors instead of lists.

$$\textsc{Return}\ \frac{(\vec{x} : \vec{X}) \in \Gamma \qquad (\boldsymbol{\alpha} : \vec{X}) \in \Delta}{\Gamma \vdash \boldsymbol{\alpha}(\vec{x}) : \Delta}$$

$$\textsc{Generator}\ \frac{f \in \mathcal{G}(\vec{X}; \vec{Y}_1, ..., \vec{Y}_\ell) \qquad (\vec{x} : \vec{X}) \in \Gamma \qquad \{(\vec{y}_i : \vec{Y}_i), \Gamma \vdash p_i : \Delta\}_{i=1}^{\ell}}{\Gamma \vdash f(\vec{x})\{\vec{y}_i \Rightarrow p_i\}_{i=1}^{\ell}}$$

$$\textsc{Loop}\ \frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad (\vec{u} : \vec{X}), \Gamma \vdash p : (\boldsymbol{\alpha} : \vec{X}), \Delta}{\Gamma \vdash \mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u} \Rightarrow p\} : \Delta}$$

A.1 Alpha equivalence

We work up to α-equivalence of variables and labels, formalized by *nominal techniques* and *variable permutations* [GP99, GP02, Cro12]: essentially, the groups of automorphisms of both variables and labels, $\mathrm{Aut}(\mathbf{V})$ and $\mathrm{Aut}(\mathbf{A})$, act on terms by structural induction (Theorems 91 and 92) and bound variables are quotiented accordingly (Theorem 94). Because we ask the sets of variables and labels, $\mathbf{V}$ and $\mathbf{A}$, to be countably infinite sets—and because any term contains always a finite number of variables and labels—there are always variables and labels that do not appear in any finite collection of terms: these are called *fresh*.

Definition 91 (Label automorphisms on terms). Automorphisms of labels, $\tau \in \mathrm{Aut}(\mathbf{A})$, act on a term, t, yielding a new term, $\tau \cdot t$, inductively defined as follows.

$$\begin{aligned} \tau \cdot (\boldsymbol{\alpha}(\vec{x})) &= (\tau\boldsymbol{\alpha})(\vec{x}); \\ \tau \cdot (\mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\}) &= \mathbf{loop}\ (\tau\boldsymbol{\alpha})(\vec{x})\{\vec{u}.(\tau \cdot p)\}; \\ \tau \cdot (f(\vec{x})\{\vec{y}_i.p_i\}) &= f(\vec{x})\{\vec{y}_i.(\tau \cdot p_i)\}. \end{aligned}$$

Definition 92 (Variable automorphisms on indexed terms). Automorphisms of variables, $\sigma \in \mathrm{Aut}(\mathbf{V})$ act on a term, p, under an index, yielding a new term, $\sigma \cdot p$, inductively defined as follows.

$$\begin{aligned} \sigma \cdot (\boldsymbol{\alpha}(\vec{x})) &= \boldsymbol{\alpha}(\sigma\vec{x}); \\ \sigma \cdot (\mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\}) &= \mathbf{loop}\ \boldsymbol{\alpha}(\sigma\vec{x})\{\sigma\vec{u}.(\sigma \cdot p)\}; \\ \sigma \cdot (f(\vec{x})\{\vec{y}_i.p_i\}) &= f(\sigma\vec{x})\{\sigma\vec{y}_i.(\sigma \cdot p_i)\}. \end{aligned}$$

Note how automorphisms act on both bound and free variables; the distinction between bound and free variables only becomes apparent when discussing alpha-equivalence (Theorem 94).

Remark 93 (Simple permutations, and shadowing). From now on, we write $(x\ y)$ to refer to the permutation that exchanges x by y and viceversa. We also write $(\vec{u}\ \vec{x})$ for the composite permutation $(u_n\ x_n) \dots (u_1\ x_1)$. Importantly for shadowing, this is different from $(u_1\ x_1) \dots (u_n\ x_n)$: while both permutations coincide whenever the variables are different, the first permutation decides that u_i will shadow u_j whenever $i < j$ for $x_i = x_j$.

Axiom 94 (Alpha-equivalence of terms). Two terms, under the same context and index, $\Gamma \vdash p : \Delta$ and $\Gamma \vdash q : \Delta$, are *α-equivalent* when they are related inductively by the following rules.

$$\textsc{Return}\ \frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad (\boldsymbol{\alpha} : X_1, ..., X_n) \in \Delta}{\Gamma \vdash \boldsymbol{\alpha}(x_1, ..., x_n)\ \equiv\ \boldsymbol{\alpha}(x_1, ..., x_n) : \Delta}$$

Loop

$$\frac{\{(\vec{x} : \vec{X}) \in \Gamma\} \qquad \boldsymbol{\gamma} \text{ fresh} \qquad (\vec{y} : \vec{Y}) \text{ fresh} \qquad \Gamma \vdash ((\vec{y}\,\vec{u}) \cdot (\boldsymbol{\gamma}\,\boldsymbol{\alpha}) \cdot p) \equiv ((\vec{y}\,\vec{v}) \cdot (\boldsymbol{\gamma}\,\boldsymbol{\beta}) \cdot q) : \boldsymbol{\gamma}(X_1, ..., X_n), \Delta}{\Gamma \vdash (\mathbf{loop}\,\boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\}) \equiv (\mathbf{loop}\,\boldsymbol{\beta}(\vec{x})\{\vec{v}.q\}) : \Delta}$$

Generator (f)

$$\frac{\{(x_i : X_i) \in \Gamma\}_{i=1}^{n} \qquad \{(\vec{y}_i : \vec{Y}_i) \text{ fresh}\}_{i=1}^{n} \qquad \{\vec{y}_i : \vec{Y}_i, \Gamma \vdash ((\vec{y}_i\,\vec{u}_i) \cdot p_i) \equiv ((\vec{y}_i\,\vec{v}_i) \cdot q_i) : \Delta\}_{i=1}^{\ell}}{\Gamma \vdash f(\vec{x})\{\vec{u}_i.p_i\}_{i=1}^{\ell} \equiv f(\vec{x})\{\vec{v}_i.q_i\}_{i=1}^{\ell} : \Delta}$$

Definition 95 (Alpha-equivalence of derivations)**.** Two derivations are *α-equivalent* if, after refreshing the variables on their contexts and the labels on their indices, their terms are α-equivalent under the same context and labels. That is, we say that $(\vec{x} : \vec{X}) \vdash p : (\vec{\boldsymbol{\alpha}} : \vec{\Psi})$ and $(\vec{y} : \vec{X}) \vdash q : (\vec{\boldsymbol{\beta}} : \vec{\Psi})$ are *α-equivalent* their substitutions with fresh variables and labels coincide.

$$\vec{z} : \vec{X} \vdash (\vec{z}\,\vec{x}) \cdot ((\vec{\boldsymbol{\omega}}\,\vec{\boldsymbol{\alpha}}) \cdot p) \equiv (\vec{z}\,\vec{y}) \cdot ((\vec{\boldsymbol{\omega}}\,\vec{\boldsymbol{\beta}}) \cdot q) : (\vec{\boldsymbol{\omega}} : \vec{\Psi}).$$

Proposition 15 (Label exchange, contraction, and weakening). *Exchange, contraction, and weakening for labels are derivable.*

lblExchange
$$\frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha}_1 : \Psi_1), (\boldsymbol{\alpha}_2 : \Psi_2), \Delta_2}{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha}_2 : \Psi_2), (\boldsymbol{\alpha}_1 : \Psi_1), \Delta_2}$$

lblContraction
$$\frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha}_1 : \Psi), (\boldsymbol{\alpha}_2 : \Psi), \Delta_2}{\Gamma \vdash \mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi), \Delta_2}$$

lblWeakening
$$\frac{\Gamma \vdash p : \Delta_1, \Delta_2}{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha} : \Psi), \Delta_2}$$

Proof. In order to derive lblExchange, we proceed by structural induction on terms: *(i)* if the term is a return statement, we simply notice that membership to the set of labels has not been altered; *(ii)* if the term is a loop, we apply the induction hypothesis to the body of the loop, which, from $(\boldsymbol{\omega} : \Psi), \Delta_1, (\boldsymbol{\alpha}_1 : \Psi_1), (\boldsymbol{\alpha}_2 : \Psi_2), \Delta_2$, becomes $(\boldsymbol{\omega} : \Psi), \Delta_1, (\boldsymbol{\alpha}_2 : \Psi_2), (\boldsymbol{\alpha}_1 : \Psi_1), \Delta_2$; and *(iii)* if the term is a generator statement, we apply the induction hypothesis to each one of its branches.

In order to derive lblContraction, we proceed by structural induction on terms: *(i)* we apply $\boldsymbol{\alpha}$, whenever we find $\boldsymbol{\alpha}_1$ or $\boldsymbol{\alpha}_2$, and leave the rest of the term unchanged. We may assume that any label $\boldsymbol{\omega}$ that we find at the head of a loop is fresh.

$$\begin{aligned}
\mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(\boldsymbol{\alpha}_1(x_1, ..., x_n)) &= \boldsymbol{\alpha}(x_1, ..., x_n);\\
\mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(\boldsymbol{\alpha}_2(x_1, ..., x_n)) &= \boldsymbol{\alpha}(x_1, ..., x_n);\\
\mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(\boldsymbol{\omega}(x_1, ..., x_n)) &= \boldsymbol{\omega}(x_1, ..., x_n), \text{ for } \omega \neq \alpha_1, \omega \neq \alpha_2\\
\mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(\mathbf{loop}\,\boldsymbol{\omega}(x_1, ..., x_n)\{p\}) &= \mathbf{loop}\,\boldsymbol{\omega}(x_1, ..., x_n)\{\mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(p)\};\\
\mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(f(\vec{x})\{\vec{y}_i \Rightarrow p_i\}) &= f(\vec{x})\{\vec{y}_i \Rightarrow \mathsf{lCntr}_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2}(p_i)\}.
\end{aligned}$$

Finally, in order to derive lblWeak, we proceed by structural induction on terms: *(i)* if the term is a return statement, we simply notice that membership to the set of labels has not been altered; *(ii)* if the term is a loop, we apply the induction hypothesis to the body of the loop; and *(iii)* if the term is a generator statement, we apply the induction hypothesis to each one of its branches. □

Proposition 16 (Index tensor exchange, contraction, weakening). *Exchange, copying, and discarding for variables on the index are derivable.*

rExchange
$$\frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X_1, X_2, \Psi_2), \Delta_2}{\Gamma \vdash \mathsf{rExch}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X_2, X_1, \Psi_2), \Delta_2}$$

rCopying
$$\frac{\Gamma \vdash p : \Delta_1, (: \Psi_1, X, \Psi_2), \Delta_2}{\Gamma \vdash \mathsf{rCopy}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X, X, \Psi_2), \Delta_2}$$

rDiscarding
$$\frac{\Gamma \vdash p : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, X, \Psi_2), \Delta_2}{\Gamma \vdash \mathsf{rDisc}(p) : \Delta_1, (\boldsymbol{\alpha} : \Psi_1, \Psi_2), \Delta_2}$$

Proof. In order to derive rExchange, we proceed by structural induction on terms. We exchange two variables each time we find the right label, $\boldsymbol{\alpha}$; and we leave the rest of the term unchanged.

$$\begin{aligned}
\mathsf{rExch}(\boldsymbol{\alpha}(y_1, ..., x_1, x_2, ..., y_n)) &= \boldsymbol{\alpha}(y_1, ..., x_2, x_1, ..., y_n);\\
\mathsf{rExch}(\boldsymbol{\omega}(z_1, ..., z_m)) &= \boldsymbol{\omega}(z_1, ..., z_m), \text{ when } \boldsymbol{\omega} \neq \boldsymbol{\alpha};\\
\mathsf{rExch}(\mathbf{loop}\ \boldsymbol{\omega}(x_1, ..., x_n)\{p\}) &= \mathbf{loop}\ \boldsymbol{\omega}(x_1, ..., x_n)\{\mathsf{rExch}(p)\};\\
\mathsf{rExch}(f(\vec{x})\{\vec{y}_i \Rightarrow p_{i,j}\}) &= f(\vec{x})\{\vec{y}_i \Rightarrow \mathsf{rExch}(p_{i,j})\}.
\end{aligned}$$

In order to derive rCopying, we proceed by structural induction on terms. We return twice the variable we are duplicating; and we leave the rest of the term unchanged.

$$\begin{aligned}
\mathsf{rCopy}(\boldsymbol{\alpha}(y_1, ..., x, ..., y_n)) &= \boldsymbol{\alpha}(y_1, ..., x, x, ..., y_n);\\
\mathsf{rCopy}(\boldsymbol{\omega}(z_1, ..., z_m)) &= \boldsymbol{\omega}(z_1, ..., z_m), \text{ when } \boldsymbol{\omega} \neq \boldsymbol{\alpha};\\
\mathsf{rCopy}(\mathbf{loop}\ \omega(x_1, ..., x_n)\{p\}) &= \mathbf{loop}\ \omega(x_1, ..., x_n)\{\mathsf{rCopy}(p)\};\\
\mathsf{rCopy}(f(\vec{x})\{\vec{y}_i \Rightarrow p_{i,j}\}) &= f(\vec{x})\{\vec{y}_i \Rightarrow \mathsf{rCopy}(p_{i,j})\}.
\end{aligned}$$

In order to derive rDiscard, we proceed by structural induction on terms. We avoid returning the variable we are discarding; and we leave the rest of the term unchanged.

$$\begin{aligned}
\mathsf{rDisc}(\boldsymbol{\alpha}(y_1, ..., x, ..., y_n)) &= \boldsymbol{\alpha}(y_1, ..., ..., y_n);\\
\mathsf{rDisc}(\boldsymbol{\omega}(z_1, ..., z_m)) &= \boldsymbol{\omega}(z_1, ..., z_m), \text{ when } \boldsymbol{\omega} \neq \boldsymbol{\alpha};\\
\mathsf{rDisc}(\mathbf{loop}\ \omega(x_1, ..., x_n)\{p\}) &= \mathbf{loop}\ \omega(x_1, ..., x_n)\{\mathsf{rDisc}(p)\};\\
\mathsf{rDisc}(f(\vec{x})\{\vec{y}_i \Rightarrow p_{i,j}\}) &= f(\vec{x})\{\vec{y}_i \Rightarrow \mathsf{rDisc}(p_{i,j})\}.
\end{aligned}$$

□

Proposition 17 (Variable exchange and contraction). *Variable exchange, variable contraction, and variable weakening are derivable.*

varExchange
$$\frac{\Gamma_1, (x : X), (y : Y), \Gamma_2 \vdash p : \Delta}{\Gamma_1, (y : Y), (x : X), \Gamma_2 \vdash p : \Delta}$$

varContraction
$$\frac{\Gamma_1, (x_1 : X), (x_2 : X), \Gamma_2 \vdash p : \Delta}{\Gamma_1, (x : X), \Gamma_2 \vdash p[x_1, x_2 \setminus x, x] : \Delta}$$

varWeakening
$$\frac{\Gamma_1, \Gamma_2 \vdash p : \Delta}{\Gamma_1, (x : X), \Gamma_2 \vdash p : \Delta}$$

Proof. We derive varExchange by structural induction: *(i)* if the term is a return statement, variable membership has is not altered and it can be constructed in the same way; *(ii)* if the term is a loop, we apply the induction hypothesis to its body; and *(iii)* if the term is a generator, we apply structural induction on each one of the branches.

We derive varContraction by structural induction: *(i)* if the term is a return statement, it now contains x in place of x_1 and x_2, so it can be derived with the new context; *(ii)* if the term is a loop, we apply substitution to its variables and the induction hypothesis to its body; and *(iii)* if the term is a generator, we apply structural induction on each one of the branches.

We derive varWeakening by structural induction: the whole term is left unchanged. □

B Proofs for Section 3 (Guards, predicates and commands)

Proposition 23. *Guards form a pair of commutative monoids, and negation is an involutive homomorphism between them.*

$$\begin{gathered}
b_1 \wedge b_2 \equiv b_2 \wedge b_1; \qquad (b_1 \wedge b_2) \wedge b_3 \equiv b_1 \wedge (b_2 \wedge b_3); \qquad b \wedge \mathbf{L} \equiv b;\\
b_1 \vee b_2 \equiv b_2 \vee b_1; \qquad (b_1 \vee b_2) \vee b_3 \equiv b_1 \vee (b_2 \vee b_3); \qquad b \vee \mathbf{R} \equiv b;\\
\neg(b_1 \wedge b_2) \equiv \neg b_2 \vee \neg b_1; \qquad \neg(\neg b) \equiv b.
\end{gathered}$$

For any total guard, $\Gamma \vdash b_t : \Omega$, we additionally have the annihilator rules, $b_t \wedge \mathsf{R} \equiv \mathsf{R}$ and $b_t \vee \mathsf{L} \equiv \mathsf{L}$. For any deterministic guard, $\Gamma \vdash b_d : \Omega$, we additionally have the idempotency rules. $b_d \wedge b_d \equiv b_d$ and $b_d \vee b_d \equiv b_d$.

Proof. Let us prove $b_1 \wedge b_2 \equiv b_2 \wedge b_1$. We reason by *(i)* the definition of conjunction, *(ii)* the interchange axiom, and *(iii)* the definition of conjunction.

$$\begin{aligned} & b_1 \wedge b_2 && \overset{(i)}{\equiv} \\ & b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2, b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_2]] && \overset{(ii)}{\equiv} \\ & b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_1, b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_2]] && \overset{(iii)}{\equiv} \\ & b_1 \wedge b_2. \end{aligned}$$

Proving $b_1 \vee b_2 \equiv b_2 \vee b_1$ is analogous.

Let us prove $\neg(b_1 \wedge b_2) \equiv \neg b_2 \wedge \neg b_1$. We reason by *(i)* definition of conjunction and negation, *(ii)* the identity substitution, *(iii)* composing substitutions, *(iv)* the definition of negation, again, *(v)* the definition of negation, and *(vi)* the definition of disjunction.

$$\begin{aligned} & \neg(b_1 \wedge b_2) && \overset{(i)}{\equiv} \\ & b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2, b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_2]][\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_1] && \overset{(ii)}{\equiv} \\ & b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2], b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_2]][\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_1] && \overset{(iii)}{\equiv} \\ & b_1[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_1], b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_1]] && \overset{(iv)}{\equiv} \\ & (\neg b_1)[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_1], b_2[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_1]] && \overset{(v)}{\equiv} \\ & (\neg b_1)[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus (\neg b_2)[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_1], (\neg b_2)[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2]] && \overset{(vi)}{\equiv} \\ & \neg b_1 \vee \neg b_2. \end{aligned}$$

The rest of the proofs are analogous. □

Proposition 26. *The following equations hold for predicate combinators: predicates form a commutative monoid with conjunction and truth, with falsehood as an absorbing element, that distributes over choices.*

$$p \wedge q \equiv q \wedge p; \qquad p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r; \qquad p \wedge \top \equiv p; \qquad p \wedge \bot \equiv \bot;$$
$$p \wedge (q +_b r) \equiv (p \wedge q) +_b (p \wedge r).$$

For any total predicate, $\Gamma \vdash p_t : \Upsilon$, we have it collapse, $p \equiv \top$. For any deterministic predicate, $\Gamma \vdash p_d : \Upsilon$, we have the idempotency rule, $p_d \wedge p_d \equiv p_d$.

Proof. Let us prove, for instance, that $p \wedge (q +_b r) \equiv (p \wedge q) +_b (p \wedge r)$. We reason by *(i)* the definition of conjunction, *(ii)* the definition of conditional, *(iii)* the interchange axiom, and *(iv)* the definitions of conditional and conjunction again.

$$\begin{aligned} & p \wedge q +_b r && \overset{(i)}{\equiv} \\ & p[\boldsymbol{\nu} \setminus q +_b r] && \overset{(ii)}{\equiv} \\ & p[\boldsymbol{\nu} \setminus b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus q, r]] && \overset{(iii)}{\equiv} \\ & b[\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \setminus p[\boldsymbol{\nu} \setminus q], p[\boldsymbol{\nu} \setminus r]] && \overset{(iv)}{\equiv} \\ & (p \wedge q) +_b (p \wedge r). \end{aligned}$$

The rest of the proofs are analogous and follow from computing substitutions. □

Proposition 29. *The following equations hold for command combinators. In particular, commands form a monoid, with composition and* skip.

$$(c_1 \mathbin{;} c_2) \mathbin{;} c_3 \equiv c_1 \mathbin{;} (c_2 \mathbin{;} c_3); \qquad (c \mathbin{;} \mathsf{skip}) \equiv c \equiv (\mathsf{skip}\mathbin{;} c); \qquad \mathsf{abort}; c \equiv \mathsf{abort} \equiv c \mathbin{;} \mathsf{abort};$$
$$\mathsf{if}\ \mathbf{L}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \equiv c_1; \qquad \mathsf{if}\ \mathbf{R}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \equiv c_2; \qquad \mathsf{if}(\neg b)\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \equiv \mathsf{if}\ b\ \mathsf{then}\ c_2\ \mathsf{else}\ c_1;$$
$$\mathsf{while}\ b\ \mathsf{do}\ c \equiv \mathsf{if}\ b\ \mathsf{then}(c \mathbin{;} \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}; \qquad \mathsf{while}\ b\ \mathsf{do}\ \mathsf{abort} \equiv \mathsf{assert}\ (\neg b)^{\#};$$
$$\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mathbin{;} d \equiv \mathsf{if}\ b\ \mathsf{then}(c_1; d)\ \mathsf{else}(c_2; d);$$
$$\mathsf{assert}\ p; \mathsf{assert}\ q \equiv \mathsf{assert}(p \wedge q); \qquad \mathsf{assert}\ b^{\#} \equiv \mathsf{if}\ b\ \mathsf{then}\ \mathsf{skip}\ \mathsf{else}\ \mathsf{abort};$$
$$\mathsf{assert} \top \equiv \mathsf{skip}; \qquad \mathsf{assert} \bot \equiv \mathsf{abort}; \qquad \mathsf{assert}(p +_b q) = \mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ p)\ \mathsf{else}(\mathsf{assert}\ q)$$

Proof. Let us prove, for instance, that $\mathsf{while}\ b\ \mathsf{do}\ c \equiv \mathsf{if}\ b\ \mathsf{then}(c \mathbin{;} \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}$. We reason by *(i)* the definition of while, *(ii)* the fixpoint rule (Theorem 14), *(iii)* the definition of while, and *(iv)* the definition of command concatenation.

$$\begin{aligned}
&\mathsf{while}\ b\ \mathsf{do}\ c && \overset{(i)}{\equiv} \\
&\mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\mathsf{if}\ b\ \mathsf{then}\ c[\boldsymbol{\eta} \setminus \vec{x}.\boldsymbol{\alpha}(\vec{x})]\ \mathsf{else}\ \mathsf{skip}\} && \overset{(ii)}{\equiv} \\
&\mathsf{if}\ b\ \mathsf{then}\ c[\boldsymbol{\eta} \setminus \vec{x}.\mathbf{loop}\ \boldsymbol{\alpha}(\vec{x})\{\mathsf{if}\ b\ \mathsf{then}\ c[\boldsymbol{\eta} \setminus \vec{x}.\boldsymbol{\alpha}(\vec{x})]\ \mathsf{else}\ \mathsf{skip}\}]\ \mathsf{else}\ \mathsf{skip} && \overset{(iii)}{\equiv} \\
&\mathsf{if}\ b\ \mathsf{then}\ c[\boldsymbol{\eta} \setminus \vec{x}.\ \mathsf{while}\ b\ \mathsf{do}\ c]\ \mathsf{else}\ \mathsf{skip} && \overset{(iv)}{\equiv} \\
&\mathsf{if}\ b\ \mathsf{then}(c; \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}\,.
\end{aligned}$$

The rest of the equations follow from similar principles. □

Lemma 96.

$$\frac{l \mathbin{;} (\!|b_1|\!)\{c_1\}\{c_2\} \le (\!|b_2|\!)\{d_1 \mathbin{;} l\}\{d_2\}}{l \mathbin{;} (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \mathbin{;} c_2 \le (\mathsf{while}\ b_2\ \mathsf{do}\ d_1) \mathbin{;} d_2}$$

Proposition 97. *The following equations hold for deterministic guards.*

$$(\!|b|\!)\{\mathsf{skip}\}\{\mathsf{skip}\} \equiv (\!|b|\!)\{\mathsf{assert}\ b^{\#}\}\{\mathsf{assert}\ (\neg b)^{\#}\}$$
$$\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \equiv \mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ b^{\#} \mathbin{;} c_1)\ \mathsf{else}(\mathsf{assert}\ (\neg b)^{\#} \mathbin{;} c_2).$$

Lemma 98. *For a total guard $b \colon X \to 1 + 1$, then* $\mathsf{if}\ b_t\ \mathsf{then}\ \mathsf{skip}\ \mathsf{else}\ \mathsf{skip} \equiv \mathsf{skip}$.

Lemma 99. *In a commutative imperative category, predicates and guards interchange: for a predicate $p \colon X \to 1$ and a guard $b \colon X \to 1 + 1$, then* $\mathsf{assert}\ p \mathbin{;} (\!|b|\!)\{\mathsf{skip}\}\{\mathsf{skip}\} = (\!|b|\!)\{\mathsf{assert}\ p\}\{\mathsf{assert}\ p\}$.

Lemma 100. *In a commutative imperative category, constant guards interchange with anything: for a guard $b \colon 1 \to 1 + 1$ and a morphism $f \colon X \to Y$, then $f \mathbin{;} (\!|b_Y|\!)\{\mathsf{skip}\}\{\mathsf{skip}\} = (\!|b_X|\!)\{f\}\{f\}$, where $b_X = \varepsilon_X \mathbin{⨾} b$ is the guard on X associated to b.*

C Proofs for Section 4 (Categorical semantics)

Definition 101 (Sesquifunctor). A (two-variable) *sesquifunctor*, $F \colon (\mathbb{A}, \mathbb{B}) \to \mathbb{C}$, consists of an assignment on objects, $F(A, B) \in \mathbb{C}_{obj}$ for $A \in \mathbb{A}_{obj}$ and $B \in \mathbb{B}_{obj}$, and two assignments on morphisms,

$$F(f; \mathrm{id}_B) \colon F(A; B) \to F(A'; B), \text{ for each } f \colon A \to A'; \text{ and}$$
$$F(\mathrm{id}_A; g) \colon F(A; B) \to F(A; B'), \text{ for each } g \colon B \to B';$$

satisfying the sesquifunctoriality axioms,

(1) $F(f \mathbin{⨾} f'; \mathrm{id}_B) = F(f, \mathrm{id}_B) \mathbin{⨾} F(f'; \mathrm{id}_B)$,
(2) $F(\mathrm{id}_A; g \mathbin{⨾} g') = F(\mathrm{id}_A, g) \mathbin{⨾} F(\mathrm{id}_A; g')$, and

(3) $F(\mathrm{id}_A;\mathrm{id}_B) = \mathrm{id}_{A\otimes B}$.

Crucially, a sesquifunctor does not necessarily satisfy the *bifunctoriality* axiom,

$$F(f;\mathrm{id}_B) \mathbin{⨟} F(\mathrm{id}_{A'};g) \neq F(\mathrm{id}_A;g) \mathbin{⨟} F(f;\mathrm{id}_{B'}).$$

Definition 102 (Symmetric premonoidal category). A *symmetric premonoidal category*—precisely, a symmetric strict premonoidal category, or permutative premonoidal category—consists of a (strict) premonoidal category endowed with a family of morphisms, $\sigma_{A,B} \colon A \otimes B \to B \otimes A$, satisfying all formal distinctly typed equations.

Lemma 47 (Terms form a predistributive copy-discard multicategory). *Terms form a predistributive copy-discard multicategory. Variable multiwhiskering (*MULTIWHISK-R *and* MULTIWHISK-L*), where we add the same type to the premises and to each one of the conclusions, are derivable.*

MULTIWHISK-L
$$\frac{\Gamma \vdash p : (\boldsymbol{\alpha}_1 : \Psi_1), \ldots, (\boldsymbol{\alpha}_n : \Psi_n)}{\Gamma, (w : X) \vdash X \ltimes p : (\boldsymbol{\alpha}_1 : X, \Psi_1), \ldots, (\boldsymbol{\alpha}_n : X, \Psi_n)}$$

MULTIWHISK-R
$$\frac{\Gamma \vdash p : (\boldsymbol{\alpha}_1 : \Psi_1), \ldots, (\boldsymbol{\alpha}_n : \Psi_n)}{\Gamma, (w : X) \vdash p \rtimes X : (\boldsymbol{\alpha}_1 : \Psi_1, X), \ldots, (\boldsymbol{\alpha}_n : \Psi_n, X)}$$

The copy-discard category structure follows from the rest of the structural rules (Theorem 16).

Proof. In order to derive MULTIWHISK-R, we proceed by structural induction on the term: *(i)* if the term is a return statement, we add the extra variable; *(ii)* if the term is a loop, we apply the induction hypothesis to the body of the loop; *(iii)* if the term is a generator, we apply the induction hypothesis to each one of its branches. In order to derive WHISKERING, we first apply MULTIWHISKERING and then RDISCARDING.

$$\boldsymbol{\alpha}(\vec{x}) \rtimes X \equiv \boldsymbol{\alpha}(\vec{x}, w);$$
$$\texttt{loop}\ \boldsymbol{\alpha}(\vec{x})\{\vec{u}.p\} \rtimes X \equiv \texttt{loop}\ \boldsymbol{\alpha}(\vec{x}, w)\{\vec{u}, v.p \rtimes X[w \setminus v]\};$$
$$(f(\vec{x})\{\vec{u}_i.p_i\}_i) \rtimes X \equiv f(\vec{x})\{\vec{u}_i.(p_i \rtimes X)\}.$$

In order to derive MULTIWHISK-L, we can use MULTIWHISK-R and the variable exchange rule (Theorem 16). □

Remark 103. Variable whiskering (whisk), where we add the same type to the premises and to one of the conclusions, is also derivable by weakening.

WHISKERING
$$\frac{\Gamma \vdash p : (\boldsymbol{\alpha}_1 : \Psi_1), \ldots, (\boldsymbol{\alpha}_n : \Psi_n)}{\Gamma, (x : X) \vdash \mathsf{whisk}(p) : (\boldsymbol{\alpha}_1 : \Psi_1, X), (\boldsymbol{\alpha}_2 : \Psi_2), \ldots, (\boldsymbol{\alpha}_n : \Psi_n)}$$

Theorem 54 (Denotational semantics). *Consider an assignment from a distributive signature $(\mathcal{B}, \mathcal{G})$ to the underlying distributive signature of an imperative multicategory, $(\mathbb{C}_{obj}, \mathbb{C})$, given by an assignment on objects, $(\!|\bullet|\!)_{obj} \colon \mathcal{B} \to \mathbb{C}_{obj}$—which extends to an assignment on lists of types, $[\![\bullet]\!]^{\otimes} \colon \mathsf{List}(\mathcal{B}) \to \mathbb{C}_{obj}$, defined inductively by $[\![]\!]^{\otimes} = I$ and $[\![X, \vec{X}]\!]^{\otimes} = [\![X]\!] \otimes [\![\vec{X}]\!]^{\otimes}$—and an assignment on generators preserving their type,*

$$(\!|\bullet|\!) \colon \mathcal{G}(\vec{X}; \vec{Y}_1, \ldots, \vec{Y}_n) \to \mathbb{C}((\!|\vec{X}|\!); (\!|\vec{Y}_1|\!) + \ldots + (\!|\vec{Y}_n|\!)).$$

It extends to an assignment, $[\![\bullet]\!] \colon (\vec{x} : \vec{X} \vdash (\boldsymbol{\alpha}_1 : \vec{Y}_1), \ldots, (\boldsymbol{\alpha}_1 : \vec{Y}_n)) \to \mathbb{C}([\![\vec{X}]\!]^{\otimes}; [\![\vec{Y}_1]\!]^{\otimes} + \ldots + [\![\vec{Y}_n]\!]^{\otimes})$, from terms to morphisms of the multicategory $\mathbb{C}$.

Proof. Let context and index be $\Gamma = (x_1 : X_1, \ldots, x_n : X_n)$ and $\Delta = (\boldsymbol{\alpha}_1 : (Y_1, \ldots, Y_{k_1})), \ldots, (\boldsymbol{\alpha}_l : (Y_1, \ldots, Y_{k_l}))$. We proceed by structural induction on terms.

Let us define the interpretation of the RETURN statement. Given any finite function $\sigma \colon m \to n$, we write $\vec{x}_\sigma$ for the list of m variables that we pick according to the function, $\vec{x}_\sigma = x_{\sigma(1)}, \ldots, x_{\sigma(m)}$. Recall that, in any copy-discard category, we have a morphism $\sigma^\star \in \mathbb{C}(X_1 \otimes ... \otimes X_n; X_{\sigma(1)} \otimes ... \otimes X_{\sigma(n)})$. Recall, moreover, that in any cocartesian multicategory, given any index i, we have an action $(\bullet) \cdot i^* : \mathbb{C}(A; B) \to \mathbb{C}(A; C_1, ..., B^{(i)}, ..., C_l)$. We define the interpretation of a RETURN statement as follows.

$$[\![\Gamma \vdash \boldsymbol{\alpha}_i(\vec{x}_\sigma) : \Delta]\!] = (\sigma^\star) \cdot i^*.$$

Let us define the interpretation of the LOOP statement. The difficulty of this case is that we want to allow two classes of variables: those that get updated by the loop and those that do not. Categorically, there is no such distinction, and all variables must be copied to each iteration of the loop to be discarded at the end. Given two finite functions, $\sigma : m_1 \to n$ and $\tau : m_2 \to n$, we write their *copairing*—the function that acts as σ on the first m_1 elements and as τ on the last m_2—as $[\sigma, \tau] : m_1 + m_2 \to n$. In the following formula, the morphism $[\sigma, \mathrm{id}_n]^\star : X_1 \otimes ... \otimes X_n \to X_{\sigma(1)} \otimes ... \otimes X_{\sigma(m)} \otimes X_1 \otimes ... \otimes X_n$ picks apart the variables that will be updated by the body of the loop; the morphism $\nu = (\mathrm{id}_m + [\mathrm{id}_n, \mathrm{id}_n])^\star : X_{\sigma(1)} \otimes ... \otimes X_{\sigma(m)} \otimes X_1 \otimes ... \otimes X_n \to X_{\sigma(1)} \otimes ... \otimes X_{\sigma(m)} \otimes X_1 \otimes ... \otimes X_n \otimes X_1 \otimes ... \otimes X_n$ passes a copy of the non-updated variables to the next iteration; and the inclusions $i_{k_j} : k_j \to k_j + n$ are used as $i^\star_{k_j} : Y_1 \otimes ... \otimes Y_{k_j} \otimes X_1 \otimes ... \otimes X_n \to Y_1 \otimes ... \otimes Y_{k_j}$ to project the relevant variables. We define the interpretation of a LOOP statement as follows.

$$[\![\Gamma \vdash \textbf{loop}\ \boldsymbol{\alpha}(\vec{x}_\sigma)\{\vec{u}.p\} : \Delta]\!] = [\sigma, \mathrm{id}_n]^\star \mathbin{⨾} \mathrm{fix}(\nu \mathbin{⨾} ([\![\vec{u} : \vec{X}_\sigma, \Gamma \vdash p : \Delta]\!] \otimes \mathrm{id}_n)) \mathbin{⨾} (i^\star_{k_1}, ..., i^\star_{k_l}).$$

Let us define the interpretation a GENERATOR statement, where we are given a generator of the form $f \in \mathcal{G}(\vec{X}; \vec{Y}_1, ..., \vec{Y}_\ell)$. Given a list of finite functions, $\sigma_1 : m_1 \to n, ..., \sigma_l : m_l \to n$, we write $[\sigma_1, ..., \sigma_l] : m_1 + ... + m_l \to n$ for its pairing. In the following formula, $\nu = [\mathrm{id}_n, \mathrm{id}_n]^\star$ copies the input and $(\bullet) \cdot [\mathrm{id}_l, ..., \mathrm{id}_l]^*$ merges the ℓ groups of outputs into a single one. We define the interpretation of a GENERATOR statement as follows.

$$[\![\Gamma \vdash f(\vec{x})\{\vec{y}_i.p_i\}_i : \Delta]\!] = (\nu \mathbin{⨾} ((\!|f|\!) \otimes \mathrm{id}_n)) \mathbin{⨾} ([\![\vec{y}_1 {:} \vec{Y}_1, \Gamma \vdash p_1 : \Delta_1]\!], ..., [\![\vec{y}_\ell {:} \vec{Y}_\ell, \Gamma \vdash p_\ell : \Delta_\ell]\!]) \cdot [\mathrm{id}_l, ..., \mathrm{id}_l]^*.$$

We provide auxiliary string diagrams in Figure 1. □

Theorem 56 (Soundness and completeness). *The denotational semantics is sound and complete for imperative multicategories.*

PROOF SKETCH. Regarding soundness, it remains to show that the definition in Theorem 54 is well-defined with respect to the axioms of the language: interchange and loop axioms in Section 2.4. Fortunately, the axioms have been chosen so as to correspond to existing axioms of traced distributive copy-discard multicategories. Indeed, the language's interchange axiom has been picked to reflect the interchange axiom of distributive multicategories; and the loop axioms (DINATURALITY, DIAGONAL, UNIFORMITY) have been picked to reflect the axioms of the trace. It only remains to formally track this correspondence by structural induction in the rules.

Regarding completeness, we have been building the syntactic model of the theory as we have been introducing the structure. We have already shown that terms form a multicategory (Theorem 41), that it is a cocartesian multicategory (Theorem 44), and that it is a predistributive copy-discard category (Theorem 47). This syntactic model means that any equation that holds for any traced distributive copy-discard multicategory holds for the syntax. □

Definition 104 (Posetal distributive copy-discard category). A *posetal distributive copy-discard category* is a distributive copy-discard category where every hom-set has a poset structure compatible with composition, tensors and coproducts: for all $f, f' : X \to Y$, $g, g' : Y \to Z$ and $h, h' : V \to W$, if $f \leq f'$, $g \leq g'$ and $h \leq h'$, then $f \mathbin{⨾} g \leq f' \mathbin{⨾} g'$, $f \otimes h \leq f' \otimes h'$ and $f + h \leq f' + h'$.

Definition 105 (Posetal uniform trace, cf. Hasegawa [Has02]). A *posetal uniform traced monoidal category* is a traced monoidal category $(\mathbb{C}, \oplus, Z)$ whose underlying monoidal category is posetally-enriched and whose trace, additionally, satisfies the *posetal uniformity axiom*: the existence of $u \colon U \to V$ such that $f ⨟ (u \oplus \mathrm{id}_Y) \leq (u \oplus \mathrm{id}_X) ⨟ g$ implies that $\mathrm{tr}(f) \leq \mathrm{tr}(g)$, for any $f \colon U \oplus X \to U \oplus Y$ and $g \colon V \oplus X \to V \oplus Y$; similarly, the existence of $v \colon V \to U$ such that $(v \oplus \mathrm{id}_X) ⨟ f \leq g ⨟ (v \oplus \mathrm{id}_Y)$ implies that $\mathrm{tr}(f) \leq \mathrm{tr}(g)$.

Definition 106 (Posetal imperative category). A *posetal imperative category* is a posetal distributive copy-discard category whose coproduct has a posetal uniform trace.

Definition 107 (Copy-discard coproducts). A copy-discard category has *copy-discard coproducts* if it has coproducts and the coproduct injections are total and deterministic. We will denote unbiased finite coproducts with $\sum$, binary coproducts with $+$ and the initial object with 0.

Definition 108 (Distributive monoidal category). A *distributive monoidal category* is a finitely-cocomplete monoidal category such that the canonical morphisms $\delta^{-\mathrm{L}}_{X;Y_1,\ldots Y_n} \colon \sum_{i=1}^n X \otimes Y_i \to X \otimes \sum_{i=1}^n Y_i$ and $\delta^{-\mathrm{R}}_{X_1,\ldots X_n;Y} \colon \sum_{i=1}^n X_i \otimes Y \to \left(\sum_{i=1}^n X_i\right) \otimes Y$ are isomorphisms.

Definition 109 (Distributive copy-discard category). A *distributive copy-discard category* is a copy-discard category $(\mathbb{C}, \otimes, I)$ with chosen finite copy-discard coproducts such that the canonical distributors

$$\delta^{-L}_{X;Y_1,\ldots Y_n} \colon \textstyle\sum_{i=1}^n X \otimes Y_i \to X \otimes \sum_{i=1}^n Y_i, \quad \text{and} \quad \delta^{-R}_{X_1,\ldots X_n;Y} \colon \sum_{i=1}^n X_i \otimes Y \to \left(\sum_{i=1}^n X_i\right) \otimes Y,$$

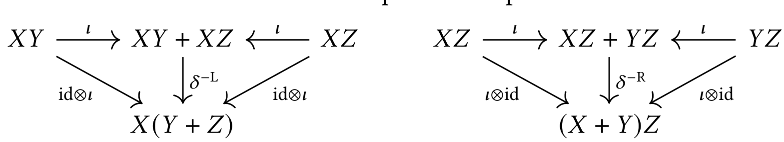
are natural isomorphisms. In particular, there are binary distributors,

$$\delta^{L}_{X;Y,Z} \colon X \otimes (Y + Z) \to X \otimes Y + X \otimes Z \quad \text{and} \quad \delta^{R}_{X,Y;Z} \colon (X + Y) \otimes Z \to X \otimes Z + Y \otimes Z.$$

Lemma 110. *The following holds in any distributive category.*

$$\iota_{XX} ⨟ (\iota_{XX} + \iota_{YY}) ⨟ (\delta^{-\mathrm{L}}_{X;X,Y} + \delta^{-\mathrm{L}}_{Y;X,Y}) ⨟ \delta^{-\mathrm{R}}_{X,Y;X+Y} = \iota_X \otimes \iota_X$$

Proof. The distributors are the canonical coproduct maps below.



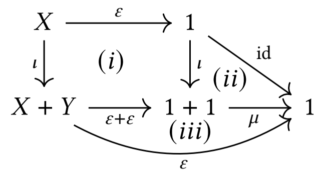
We rewrite the left-hand side using (5, 8) that the distributors are the canonical ones, (6, 9) the properties of coproducts, and (7) naturality of injections.

$$\begin{aligned}
&\iota_{XX} ⨟ (\iota_{XX} + \iota_{YY}) ⨟ (\delta^{-\mathrm{L}}_{X;X,Y} + \delta^{-\mathrm{L}}_{Y;X,Y}) ⨟ \delta^{-\mathrm{R}}_{X,Y;X+Y} \\
&= \iota_{XX} ⨟ ((\iota_{XX} ⨟ [\mathrm{id}_X \otimes \iota_X, \mathrm{id}_X \otimes \iota_Y]) + (\iota_{YY} ⨟ [\mathrm{id}_Y \otimes \iota_X, \mathrm{id}_Y \otimes \iota_Y])) ⨟ \delta^{-\mathrm{R}}_{X,Y;X+Y} && (5) \\
&= \iota_{XX} ⨟ ((\mathrm{id}_X \otimes \iota_X) + (\mathrm{id}_Y \otimes \iota_Y)) ⨟ \delta^{-\mathrm{R}}_{X,Y;X+Y} && (6) \\
&= (\mathrm{id}_X \otimes \iota_X) ⨟ \iota_{X(X+Y)} ⨟ \delta^{-\mathrm{R}}_{X,Y;X+Y} && (7) \\
&= (\mathrm{id}_X \otimes \iota_X) ⨟ \iota_{X(X+Y)} ⨟ [\iota_X \otimes \mathrm{id}_{X+Y}, \iota_Y \otimes \mathrm{id}_{X+Y}] && (8) \\
&= (\mathrm{id}_X \otimes \iota_X) ⨟ (\iota_X \otimes \mathrm{id}_{X+Y}) && (9) \\
&= \iota_X \otimes \iota_X.
\end{aligned}$$

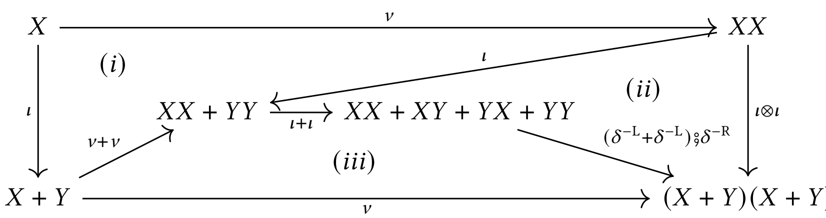
This concludes the proof. □

Proposition 111. *Let $\mathbb{C}$ be a copy-discard category that is also distributive monoidal. Then, it is a distributive copy-discard category if and only if the copy and discard morphisms are compatible with coproducts, $\nu_{X+Y} = (\nu_X + \zeta_{X \otimes Y} + \zeta_{Y \otimes X} + \nu_Y) ⨟ (\delta^{-\mathrm{L}}_{X;X,Y} + \delta^{-\mathrm{L}}_{Y;X,Y}) ⨟ \delta^{-\mathrm{R}}_{X,Y;X+Y}$ and $\varepsilon_{X+Y} = (\varepsilon_X + \varepsilon_Y) ⨟ \mu_1$.*

Proof. Suppose that the copy and discard morphisms are compatible with coproducts. We show that $\iota_X ⨾ \varepsilon_{X+Y} = \varepsilon_X$, i.e. that the outer diagram below commutes.

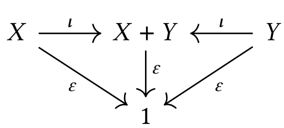


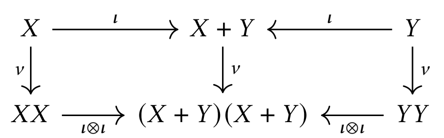
The diagram (i) commutes by naturality of the injection ι_X; the diagram (ii) commutes by unitality of the structure morphism of the coproduct μ_I; the diagram (iii) commutes by hypothesis. Similarly, we show that $\iota_X ⨾ \nu_{X+Y} = \nu_X ⨾ (\iota_X \otimes \iota_X)$, i.e. that the outer diagram below commutes. We omit the symbol $\otimes$ for the monoidal product to ease readability.



The diagram (i) commutes by naturality of the injection ι_X; the diagram (ii) commutes by Theorem 110; the diagram (iii) commutes by hypothesis.

Conversely, suppose that the coproduct injections are total and deterministic. Then, the two diagrams below commute.





By the universal property of coproducts, we must have $\varepsilon_{X+Y} = [\varepsilon_X, \varepsilon_Y] = (\varepsilon_X + \varepsilon_Y) ⨾ \mu_1$ and equation (10) below. Equations (11, 12) follow from properties of coproducts, while (13, 14) follow from the canonicity of distributors.

$$
\begin{aligned}
&\nu_{X+Y} \\
&= [\nu_X ⨾ (\iota_X \otimes \iota_X), \nu_Y ⨾ (\iota_Y \otimes \iota_Y)] && (10)\\
&= (\nu_X + \nu_Y) ⨾ [\iota_X \otimes \iota_X, \iota_Y \otimes \iota_Y] && (11)\\
&= (\nu_X + \nu_Y) ⨾ ((\mathrm{id}_X \otimes \iota_X) + (\mathrm{id}_Y \otimes \iota_Y)) ⨾ [\iota_X \otimes \mathrm{id}_{X+Y}, \iota_Y \otimes \mathrm{id}_{X+Y}] && (12)\\
&= (\nu_X + \nu_Y) ⨾ ((\mathrm{id}_X \otimes \iota_X) + (\mathrm{id}_Y \otimes \iota_Y)) ⨾ \delta^{-\mathrm{R}}_{X,Y;X+Y} && (13)\\
&= (\nu_X + \nu_Y) ⨾ ((\iota_{XX} ⨾ \delta^{-\mathrm{L}}_{X;X,Y}) + (\iota_{YY} ⨾ \delta^{-\mathrm{L}}_{Y;X,Y})) ⨾ \delta^{-\mathrm{R}}_{X,Y;X+Y} && (14)\\
&= (\nu_X + \nu_Y) ⨾ (\iota_{XX} + \iota_{YY}) ⨾ (\delta^{-\mathrm{L}}_{X;X,Y} + \delta^{-\mathrm{L}}_{Y;X,Y}) ⨾ \delta^{-\mathrm{R}}_{X,Y;X+Y} \\
&= (\nu_X + \zeta_{XY} + \zeta_{YX} + \nu_Y) ⨾ (\delta^{-\mathrm{L}}_{X;X,Y} + \delta^{-\mathrm{L}}_{Y;X,Y}) ⨾ \delta^{-\mathrm{R}}_{X,Y;X+Y}
\end{aligned}
$$

□

Lemma 62. *In a distributive copy-discard category, the structure morphisms of coproducts, μ and ζ, are total and deterministic.*

Proof. By initiality of 0, we obtain that $\nu_0 \fatsemi (\zeta_X \otimes \zeta_X) = \zeta_X \fatsemi \nu_X$ and that $\varepsilon_0 = \zeta_X \fatsemi \varepsilon_X$. By the hypothesis on the discard maps, ε, and by naturality of μ, we obtain that the maps μ are total: $\varepsilon_{X+X} = (\varepsilon_X + \varepsilon_X) \fatsemi \mu_1 = \mu_X \fatsemi \varepsilon_X$. By (15) the hypothesis on the copy maps, ν, by (16, 17) the canonicity of the distributors, by (19, 21) naturality of μ, and by (20) by properties of coproducts, we obtain that the maps μ are deterministic.

$$
\begin{aligned}
&\nu_{X+X} \fatsemi (\mu_X \otimes \mu_X) \\
&= (\nu_X + \zeta_{XX} + \zeta_{XX} + \nu_X) \fatsemi (\delta^{-\mathrm{L}}_{X;X,X} + \delta^{-\mathrm{L}}_{X;X,X}) \fatsemi \delta^{-\mathrm{R}}_{X,X;X+X} \fatsemi (\mu_X \otimes \mu_X) \\
&= (\nu_X + \nu_X) \fatsemi ((\iota_{XX} \fatsemi \delta^{-\mathrm{L}}_{X;X,X}) + (\iota_{XX} \fatsemi \delta^{-\mathrm{L}}_{X;X,X})) \fatsemi \delta^{-\mathrm{R}}_{X,X;X+X} \fatsemi (\mu_X \otimes \mu_X) && (15)\\
&= (\nu_X + \nu_X) \fatsemi ((\mathrm{id}_X \otimes \iota_X) + (\mathrm{id}_X \otimes \iota_X)) \fatsemi \delta^{-\mathrm{R}}_{X,X;X+X} \fatsemi (\mu_X \otimes \mu_X) && (16)\\
&= (\nu_X + \nu_X) \fatsemi ((\mathrm{id}_X \otimes \iota_X) + (\mathrm{id}_X \otimes \iota_X)) && (17)\\
&\qquad \fatsemi ((\iota_X \otimes \mathrm{id}_{X+X}) + (\iota_X \otimes \mathrm{id}_{X+X})) \fatsemi \mu_{(X+X)(X+X)} \fatsemi (\mu_X \otimes \mu_X) && (18)\\
&= (\nu_X + \nu_X) \fatsemi ((\iota_X \otimes \iota_X) + (\iota_X \otimes \iota_X)) \fatsemi ((\mu_X \otimes \mu_X) + (\mu_X \otimes \mu_X)) \fatsemi \mu_{XX} && (19)\\
&= (\nu_X + \nu_X) \fatsemi \mu_{XX} && (20)\\
&= \mu_X \fatsemi \nu_X && (21)
\end{aligned}
$$

□

Remark 112 (Bimonoidally strict distributive category). A distributive category is *bimonoidally strict*—or simply *strict*, in this text—when both its monoidal and cocartesian structures are strict. Every distributive category is equivalent to a bimonoidally strict one: in fact, equivalent to one where one of the left distributor (respectively, the right distributor) is the identity [Lap06]. However, not every distributive category is equivalent to a fully strict one: if both distributors were to be identities, the following strict equality

$$AC + AD + BC + BD = (A + B)(C + D) = AC + BC + AD + BD,$$

would force the coproduct to be commutative, instead of symmetric.

Proposition 75. *Under the conditions of Theorem 73, the Kleisli category of a monad,* $\mathsf{kl}(T)$*, has a posetal uniform trace.*

Proof. We first recall the construction of the monoidal trace in Theorem 73. Hereafter, identities (e.g. $\mathrm{id}_Y \colon Y \to Y$), injections ($\kappa_U \colon U \to U + X$) and coproducts ($+$) are all in $\mathsf{kl}(T)$. Moreover, we write $\Sigma_{n\in\mathbb{N}} Y$ for the countable coproduct of an object Y and $\nabla \colon \Sigma_{n\in\mathbb{N}} Y \to Y$ for the copairing of id_Y.

For each $f \colon U + X \to U + Y$ in $\mathsf{kl}(T)$, one defines $\hat{f} \colon (U + X) \to (U + X) + Y$ as $f \fatsemi (\kappa_U + \mathrm{id}_Y)$. This is a coalgebra for the functor $Id + Y \colon \mathsf{kl}(T) \to \mathsf{kl}(T)$. One can show that $\Sigma_{n\in\mathbb{N}} Y$ carries a final coalgebra for such functor and thus one has a unique coalgebra morphism $!_{\hat{f}} \colon (U + X) \to \Sigma_{n\in\mathbb{N}} Y$. It is shown in Theorem 5.2 in [Jac10] that defining $Tr(f) \colon X \to Y$ as

$$\mathrm{tr}(f) = \kappa_X \fatsemi !_{\hat{f}} \fatsemi \nabla \tag{22}$$

provides a uniform monoidal trace.

In order to prove posetal uniformity we rely on a previous result [Has06, Proposition 5.6], stated under the same conditions of Theorem 73 but restricted to the case $\mathbb{C} = \mathbb{Set}$; one can carefully check that its proof also works for arbitrary categories $\mathbb{C}$ with countable coproducts.

Take $f \colon U + X \to U + Y$, $g \colon V + X \to V + Y$ and $u \colon U \to V$ in $\mathsf{kl}(T)$ and assume that

$$f \fatsemi (u \oplus \mathrm{id}_Y) \geq (u \oplus \mathrm{id}_X) \fatsemi g. \tag{23}$$

As for $\hat{f}\colon U \to U + Y$, we define the coalgebra $\hat{g}\colon V \to V + Y$ and consider the unique coalgebra morphim $!_{\hat{g}}\colon V \to \Sigma_{n\in\mathbb{N}} Y$. From (23), one easily derive that

$$\hat{f} \,⨾\, ((u + \mathrm{id}_X) \oplus \mathrm{id}_Y) \geq (u \oplus \mathrm{id}_X) \,⨾\, \hat{g},$$

namely, using the terminology in [Has06], $(u + \mathrm{id}_X)$ is a *lax-coalgebra morphism* from $\hat{f}$ to $\hat{g}$. Now $(u + \mathrm{id}_X)⨾!_{\hat{g}}\colon U + X \to \Sigma_{n\in\mathbb{N}} Y$ is also a lax-coalgebra morphism. By Proposition 5.6 in [Has06], the unique coalgebra morphism $!_{\hat{f}}\colon U + X \to \Sigma_{n\in\mathbb{N}} Y$ is the *greatest* lax coalgebra morphism and thus

$$!_{\hat{f}} \geq (u + \mathrm{id}_X)⨾!_{\hat{g}}. \tag{24}$$

We can then conclude with the following derivation.

$$\begin{aligned}
\mathrm{tr}(f) &= \kappa_X ⨾ !_{\hat{f}} ⨾ \nabla && (22)\\
&\geq \kappa_X ⨾ (u + \mathrm{id}_X) ⨾ !_{\hat{g}} ⨾ \nabla && (24)\\
&= \kappa_X ⨾ !_{\hat{g}} ⨾ \nabla && \text{(coproduct)}\\
&= \mathrm{tr}(g) && (22)
\end{aligned}$$

For proving the other implication, one proceeds by reversing the inequalities and use the fact that, by Proposition 5.6 in [Has06], $!_{\hat{f}}$ is the smallest *oplax* coalgebra morphism. □

Corollary 76. *The Kleisli categories of the maybe monad, powerset monad, and subdistributions monad on the distributive category* Set*, and of the subdistributions monad on the distributive category* StdBorel *are posetal imperative categories.*

Proof. For the monads on Set, the assumptions of Theorem 73 are already checked in [Jac10]. We now check the conditions for the monad $\mathcal{G}$ on StdBorel. The countable coproduct of standard Borel spaces is again standard Borel, so StdBorel has countable coproducts. The Kleisli category of $\mathcal{G}$ is poset-enriched with the pointwise order and it has a bottom element, the zero subdistribution. Moreover, hom-sets are DCPOs because the supremum of an increasing sequence of measurable functions is defined pointwise and bounded increasing sequences of real numbers have a supremum. Finally, cotuplings are monotone because they are so pointwise. □

D Proofs for Section 5 (Distributive program logics)

Theorem 79. *The following are valid assertion-correctness triples in any posetal imperative category where* abort $\leq f$ *and* $f ⨾ \top \leq \top$ *for all morphisms* f.

SKIP
$$\frac{}{\{p\}\ \mathsf{skip}\ \{p\}}$$

COMP
$$\frac{\{p\}\ c_1\ \{q\} \qquad \{q\}\ c_2\ \{r\}}{\{p\}\ c_1\,;c_2\ \{r\}}$$

ASSIGN
$$\frac{e \text{ deterministic and total}}{\{p[u \setminus e]\}\ u \coloneqq e\ \{p\}}$$

CHOICE
$$\frac{\{p\}\ c_1\ \{q\} \qquad \{p\}\ c_2\ \{q\}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}}$$

LOOP
$$\frac{\{p\}\ c\ \{p\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{p\}}$$

UNROLL
$$\frac{\{p\}\ \mathsf{if}\ b\ \mathsf{then}(c\,;\mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else\ skip}\ \{q\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{q\}}$$

IFELSE
$$\frac{\{p \wedge b^{\#}\}\ c_1\ \{q\} \qquad \{p \wedge (\neg b)^{\#}\}\ c_2\ \{q\} \qquad b \text{ deterministic}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}}$$

WHILE
$$\frac{\{b^{\#} \wedge p\}\ c\ \{p\} \qquad b \text{ deterministic}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{p \wedge (\neg b)^{\#}\}}$$

MONOTONE
$$\frac{p_1 \leq p_2 \qquad \{p_2\}\ c\ \{q_2\} \qquad q_2 \leq q_1}{\{p_1\}\ c\ \{q_1\}}$$

AND
$$\frac{\{p_1\}\ c\ \{q_1\} \qquad \{p_2\}\ c\ \{q_2\}}{\{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\}}$$

FAIL
$$\frac{}{\{p\}\ \mathsf{abort}\ \{q\}}$$

$$\textit{ASSERT}\ \frac{q \wedge r \leq \bot}{\{p +_b q\}\ \mathsf{assert}\ r\ \{p \wedge b^{\#}\}} \qquad \textit{TOP}\ \frac{}{\{p\}\ c\ \{\top\}} \qquad \textit{BOT}\ \frac{}{\{\bot\}\ c\ \{q\}}$$

Proof. The SKIP rule follows from neutrality of skip (Theorem 29) and reflexivity of the preorder.

$$\mathsf{assert}\ p \mathbin{;} \mathsf{skip} \equiv \mathsf{assert}\ p \leq \mathsf{assert}\ p \equiv \mathsf{skip} \mathbin{;} \mathsf{assert}\ p$$

The COMP rule follows from its first and second premises, implicitly using associativity of concatenation (Theorem 29) and the congruence of the preorder.

$$\mathsf{assert}\ p \mathbin{;} c_1 \mathbin{;} c_2 \leq c_1 \mathbin{;} \mathsf{assert}\ q \mathbin{;} c_2 \leq c_1 \mathbin{;} c_2 \mathbin{;} \mathsf{assert}\ r$$

The ASSIGN rule follows from the definition of expression substitution (Theorem 24), determinism of e and, implicitly, from reflexivity of the preorder.

$$\mathsf{assert}\ p[u \setminus e] \mathbin{;} (u \coloneqq e) = \mathsf{assert}((u \coloneqq e) \mathbin{;} p) \mathbin{;} (u \coloneqq e) = (u \coloneqq e) \mathbin{;} \mathsf{assert}\ p$$

The CHOICE rule follows by *(i)* Theorem 99, *(ii)* both assumptions, $\{p\}\ c_1\ \{q\}$ and $\{p\}\ c_2\ \{q\}$, and *(iii)* the definition of composition.

$$\begin{aligned}
&\mathsf{assert}\ p \mathbin{;} \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 && \overset{(i)}{\equiv}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ p \mathbin{;} c_1)\ \mathsf{else}(\mathsf{assert}\ p \mathbin{;} c_2) && \overset{(ii)}{\leq}\\
&\mathsf{if}\ b\ \mathsf{then}(c_1 \mathbin{;} \mathsf{assert}\ q)\ \mathsf{else}(c_2 \mathbin{;} \mathsf{assert}\ q) && \overset{(iii)}{\equiv}\\
&(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \mathbin{;} q.
\end{aligned}$$

The IFELSE rule follows from *(i)* determinism of b (Theorem 97), *(ii)* Theorem 99, *(iii)* the definition of predicate conjunction (Theorem 24) *(iv)* the hypotheses, and *(v)* the definition of composition of program fragments (Theorem 27).

$$\begin{aligned}
&\mathsf{assert}\ p \mathbin{;} \mathsf{if}\ b\ \mathsf{then}\ f\ \mathsf{else}\ g && \overset{(i)}{=}\\
&\mathsf{assert}\ p \mathbin{;} \mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ b^{\#} \mathbin{;} f)\ \mathsf{else}(\mathsf{assert}\ (\neg b)^{\#} \mathbin{;} g) && \overset{(ii)}{=}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ p \mathbin{;} \mathsf{assert}\ b^{\#} \mathbin{;} f)\ \mathsf{else}(\mathsf{assert}\ p \mathbin{;} \mathsf{assert}\ (\neg b)^{\#} \mathbin{;} g) && \overset{(iii)}{=}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}(p \wedge b^{\#}) \mathbin{;} f)\ \mathsf{else}(\mathsf{assert}(p \wedge (\neg b)^{\#}) \mathbin{;} g) && \overset{(iv)}{=}\\
&\mathsf{if}\ b\ \mathsf{then}(f \mathbin{;} \mathsf{assert}\ q)\ \mathsf{else}(g \mathbin{;} \mathsf{assert}\ q) && \overset{(v)}{=}\\
&(\mathsf{if}\ b\ \mathsf{then}\ f\ \mathsf{else}\ g) \mathbin{;} \mathsf{assert}\ q.
\end{aligned}$$

For the LOOP rule, we apply the uniformity principle (Theorem 29); the antecedent of the uniformity rule follows from *(i)* Theorem 99, and *(ii)* the correctness assumption.

$$\begin{aligned}
&\mathsf{assert}\ p \mathbin{;} (\!|b|\!)\{c\}\{\mathsf{skip}\} && \overset{(i)}{=}\\
&(\!|b|\!)\{\mathsf{assert}\ p \mathbin{;} c\}\{\mathsf{assert}\ p\} && \overset{(ii)}{\leq}\\
&(\!|b|\!)\{c \mathbin{;} \mathsf{assert}\ p\}\{\mathsf{assert}\ p\}
\end{aligned}$$

Then, by uniformity, $\mathsf{assert}\ p \mathbin{;} \mathsf{while}\ b\ \mathsf{do}\ c = \mathsf{assert}\ p \mathbin{;} \mathsf{while}\ b\ \mathsf{do}\ c \mathbin{;} \mathsf{skip} \leq \mathsf{while}\ b\ \mathsf{do}\ c \mathbin{;} \mathsf{assert}\ p$.

The WHILE rule is similar to the LOOP rule, but additionally uses *(ii)* determinism of b (Theorem 97).

$$\begin{aligned}
&\mathsf{assert}\ p \mathbin{;} (\!|b|\!)\{c\}\{\mathsf{skip}\} && \overset{(i)}{=}\\
&(\!|b|\!)\{\mathsf{assert}\ p \mathbin{;} c\}\{\mathsf{assert}\ p\} && \overset{(ii)}{=}\\
&(\!|b|\!)\{\mathsf{assert}\ b^{\#} \mathbin{;} \mathsf{assert}\ p \mathbin{;} c\}\{\mathsf{assert}\ (\neg b)^{\#} \mathbin{;} \mathsf{assert}\ p\} && \overset{(iii)}{=}\\
&(\!|b|\!)\{\mathsf{assert}(b^{\#} \wedge p) \mathbin{;} c\}\{\mathsf{assert}((\neg b)^{\#} \wedge p)\} && \overset{(iv)}{\leq}\\
&(\!|b|\!)\{c \mathbin{;} \mathsf{assert}\ p\}\{\mathsf{assert}((\neg b)^{\#} \wedge p)\}
\end{aligned}$$

Then, $\mathsf{assert}\ p\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c = \mathsf{assert}\ p\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c\ ;\ \mathsf{skip} \leq \mathsf{while}\ b\ \mathsf{do}\ c\ ;\ \mathsf{assert}((\neg b)^{\#} \wedge p)$.

The UNROLL rule follows from *(i)* Theorem 29 and *(ii)* the assumption.

$$\begin{aligned}
&\mathsf{assert}\ p\ ;\ (\mathsf{while}\ b\ \mathsf{do}\ c) &\overset{(i)}{\equiv}\\
&\mathsf{assert}\ p\ ;\ (\mathsf{if}\ b\ \mathsf{then}(c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}) &\overset{(ii)}{\leq}\\
&(\mathsf{if}\ b\ \mathsf{then}(c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip})\ ;\ \mathsf{assert}\ q &\overset{(i)}{\equiv}\\
&(\mathsf{while}\ b\ \mathsf{do}\ c)\ ;\ \mathsf{assert}\ q
\end{aligned}$$

The MONOTONE rule follows from monotonicity of composition.

$$\mathsf{assert}\ p_1\ ;\ c \leq \mathsf{assert}\ p_2\ ;\ c \leq c\ ;\ \mathsf{assert}\ q_2 \leq c\ ;\ \mathsf{assert}\ q_1$$

The AND rule follows from the properties of assertions (Theorem 29).

$$\begin{aligned}
\mathsf{assert}(p_1 \wedge p_2)\ ;\ c &= \mathsf{assert}\ p_1\ ;\ \mathsf{assert}\ p_2\ ;\ c \leq \mathsf{assert}\ p_1\ ;\ c\ ;\ \mathsf{assert}\ q_2\\
&\leq c\ ;\ \mathsf{assert}\ q_1\ ;\ \mathsf{assert}\ q_2 = c\ ;\ \mathsf{assert}(q_1 \wedge q_2)
\end{aligned}$$

The FAIL rule follows from the properties of abort (Theorem 29).

$$\mathsf{assert}\ p\ ;\ \mathsf{abort} = \mathsf{abort} = \mathsf{abort}\ ;\ \mathsf{assert}\ q$$

The ASSERT rule follows from *(i)* Theorem 29, *(ii)* the definition of commands composition (Theorem 27), *(iii)* the hypotheses, *(iv)* Theorem 29, *(v)* Theorem 99, *(vi)* Theorem 29, and *(vii)* Theorem 29.

$$\begin{aligned}
&\mathsf{assert}(p +_b q)\ ;\ \mathsf{assert}\ r &\overset{(i)}{=}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ p)\ \mathsf{else}(\mathsf{assert}\ q)\ ;\ \mathsf{assert}\ r &\overset{(ii)}{=}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ p\ ;\ \mathsf{assert}\ r)\ \mathsf{else}(\mathsf{assert}\ q\ ;\ \mathsf{assert}\ r) &\overset{(iii)}{\leq}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ p\ ;\ \mathsf{assert}\ r)\ \mathsf{else}(\mathsf{assert}\ \bot) &\overset{(iv)}{=}\\
&\mathsf{if}\ b\ \mathsf{then}(\mathsf{assert}\ r\ ;\ \mathsf{assert}\ p)\ \mathsf{else}(\mathsf{assert}\ r\ ;\ \mathsf{assert}\ p\ ;\ \mathsf{assert}\ \bot) &\overset{(v)}{=}\\
&\mathsf{assert}\ r\ ;\ \mathsf{assert}\ p\ ;\ \mathsf{if}\ b\ \mathsf{then}\ \mathsf{skip}\ \mathsf{else}\ \mathsf{abort} &\overset{(vi)}{=}\\
&\mathsf{assert}\ r\ ;\ \mathsf{assert}\ p\ ;\ \mathsf{assert}(b^{\#}) &\overset{(vii)}{=}\\
&\mathsf{assert}\ r\ ;\ \mathsf{assert}(p \wedge b^{\#})
\end{aligned}$$

The TOP and BOT rules follow from *(i)* the extra hypotheses, *(ii)* Theorem 29, and *(iii)* Theorem 29.

$$\begin{aligned}
&\mathsf{assert}\ p\ ;\ c &\overset{(i)}{\leq} \qquad\qquad &\mathsf{assert}\ \bot\ ;\ c &\overset{(iii)}{=}\\
&\mathsf{assert}\ \top\ ;\ c &\overset{(ii)}{=} \qquad\qquad &\mathsf{abort}\ ;c &\overset{(iii)}{=}\\
&c &\overset{(ii)}{=} \qquad\qquad &c\ ;\ \mathsf{abort} &\overset{(i)}{\leq}\\
&c\ ;\ \mathsf{assert}\ \top & \qquad\qquad &c\ ;\ \mathsf{assert}\ q
\end{aligned}$$

□

Theorem 81. *The following are valid state-incorrectness triples in any posetal imperative category where* abort $\leq f$ *for all morphisms* f.

$$\textit{SKIP}\ \frac{}{\{s\}\ \mathsf{skip}\ \{s\}} \qquad \textit{COMP}\ \frac{\{s\}\ c_1\ \{t\} \qquad \{t\}\ c_2\ \{r\}}{\{s\}\ c_1\ ;\ c_2\ \{r\}} \qquad \textit{COMP (ERROR)}\ \frac{\{s\}\ c_1\ \{\bot\}}{\{s\}\ c_1\ ;\ c_2\ \{\bot\}}$$

$$\textit{ASSIGN}\ \frac{}{\{s\}\ x \coloneqq y\ \{s(x \setminus y)\}} \qquad \textit{SAMPLE}\ \frac{}{\{s\}\ x \leftarrow s_0\ \{\textstyle\coprod_x s \cdot s_0\}}$$

$$\textit{CHOICE (LEFT)}\ \frac{\{s \downharpoonleft b^{\#}\}\ c_1\ \{t\}}{\{s\}\ \text{if } b \text{ then } c_1 \text{ else } c_2\ \{t\}} \qquad \textit{CHOICE (RIGHT)}\ \frac{\{s \downharpoonleft (\neg b)^{\#}\}\ c_2\ \{t\}}{\{s\}\ \text{if } b \text{ then } c_1 \text{ else } c_2\ \{t\}} \qquad \textit{CONVEX}\ \frac{\{s_1\}\ c\ \{t_1\} \qquad \{s_2\}\ c\ \{t_2\} \qquad b \textit{ constant}}{\{s_1 +_b s_2\}\ c\ \{t_1 +_b t_2\}}$$

$$\textit{ITER ZERO}\ \frac{}{\{s\}\ \text{while } b \text{ do } c\ \{s \downharpoonleft (\neg b)^{\#}\}} \qquad \textit{ITER}\ \frac{\{s \downharpoonleft b^{\#}\}\ c\ ;\ \text{while } b \text{ do } c\ \{t\}}{\{s\}\ \text{while } b \text{ do } c\ \{t\}}$$

$$\textit{MONOTONE}\ \frac{s_1 \geq s_2 \qquad \{s_2\}\ c\ \{t_2\} \qquad t_2 \geq t_1}{\{s_1\}\ c\ \{t_1\}} \qquad \textit{ASSERT}\ \frac{}{\{s\}\ \text{assert } p\ \{s \downharpoonleft p\}} \qquad \textit{FAIL}\ \frac{}{\{s\}\ \text{abort}\ \{\bot\}} \qquad \textit{BOT}\ \frac{}{\{s\}\ c\ \{\bot\}}$$

Proof. The skip and comp rules follow from Theorem 29. The comp (error) rule follows from naturality of abort (Theorem 29).

$$s\ ;\ \text{skip} = s \qquad s\ ;\ c_1\ ;\ c_2 \geq t\ ;\ c_2 \geq u \qquad s\ ;\ c_1\ ;\ c_2 \geq \bot\ ;\ c_2 = \bot$$

The assign and sample rules follow from the definitions of the state combinators (Theorem 31).

$$s\ ;\ (x \coloneqq y) = s(x \setminus y) \qquad s\ ;\ (x \leftarrow s_x) = \textstyle\coprod_x s \cdot s_x$$

The choice (left) and choice (right) rules follow from *(i)* the hypothesis, *(ii)* Theorem 29, *(iii)* Theorem 31, and *(iv)* the assumption.

$$\begin{aligned} & s\ ;\ (\text{if } b \text{ then } c_1 \text{ else } c_2) \overset{(i)}{\geq} \\ & s\ ;\ (\text{if } b \text{ then } c_1 \text{ else abort}) \overset{(ii)}{=} \\ & s\ ;\ \text{assert } b^{\#}\ ;\ c_1 \overset{(iii)}{=} \\ & (s \downharpoonleft b^{\#})\ ;\ c_1 \overset{(iv)}{\geq} \\ & t \end{aligned} \qquad \begin{aligned} & s\ ;\ (\text{if } b \text{ then } c_1 \text{ else } c_2) \overset{(i)}{\geq} \\ & s\ ;\ (\text{if } b \text{ then abort else } c_2) \overset{(ii)}{=} \\ & s\ ;\ \text{assert } (\neg b)^{\#}\ ;\ c_2 \overset{(iii)}{=} \\ & (s \downharpoonleft (\neg b)^{\#})\ ;\ c_2 \overset{(iv)}{\geq} \\ & t \end{aligned}$$

The convex rule follows from the definition of command composition (Theorem 27).

$$s_1 +_b s_2\ ;\ c = (s_1\ ;\ c) +_b (s_2\ ;\ c) \geq t_1 +_b t_2$$

The iter zero rule follows from *(i)* the hypothesis, *(ii)* Theorem 29 and *(iii)* Theorem 31.

$$s\ ;\ \text{while } b \text{ do } c \overset{(i)}{\geq} s\ ;\ \text{while } b \text{ do abort} \overset{(ii)}{=} s\ ;\ \text{assert } (\neg b)^{\#} \overset{(iii)}{=} s \downharpoonleft (\neg b)^{\#}$$

The iter rule follows from *(i)* Theorem 29, *(ii)* the hypothesis, *(iii)* Theorem 29, *(iv)* Theorem 31, and *(v)* the assumption.

$$\begin{aligned} & s\ ;\ \text{while } b \text{ do } c \overset{(i)}{=} \\ & s\ ;\ (\text{if } b \text{ then} (c\ ;\ \text{while } b \text{ do } c) \text{ else skip}) \overset{(ii)}{\geq} \\ & s\ ;\ (\text{if } b \text{ then} (c\ ;\ \text{while } b \text{ do } c) \text{ else abort}) \overset{(iii)}{=} \\ & s\ ;\ \text{assert } b^{\#}\ ;\ c\ ;\ \text{while } b \text{ do } c \overset{(iv)}{=} \\ & (s \downharpoonleft b^{\#})\ ;\ c\ ;\ \text{while } b \text{ do } c \overset{(v)}{\geq} \\ & t \end{aligned}$$

The monotone rule follows from monotonicity of command composition. The assert rule applies Theorem 31. The fail rule follows from Theorem 29. The bot rule follows from the hypothesis.

$$s_1\ ;\ c \geq s_2\ ;\ c \geq t_2 \geq t_1 \qquad s \fatsemi \text{assert } p = s \downharpoonleft p \qquad s\ ;\ \text{abort} = \bot \qquad s\ ;\ c \geq \bot$$

□

Theorem 83. *The following are valid predicate-correctness triples in any posetal imperative category where* abort $\leq f$ *for all morphisms* f.

$$\textit{SKIP}\ \frac{}{\{p\}\ \mathsf{skip}\ \{p\}} \qquad \textit{COMP}\ \frac{\{p\}\ c_1\ \{q\} \qquad \{q\}\ c_2\ \{r\}}{\{p\}\ c_1\ ;c_2\ \{r\}} \qquad \textit{ASSIGN}\ \frac{e \text{ deterministic}}{\{p[u \setminus e]\}\ u \coloneqq e\ \{p\}} \qquad \textit{SAMPLE}\ \frac{}{\{p[u \setminus s]\}\ u \leftarrow s\ \{p\}}$$

$$\textit{UNROLL}\ \frac{\{p\}\ \mathsf{if}\ b\ \mathsf{then}(c\ ;\mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}\ \{q\}}{\{p\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{q\}} \qquad \textit{CHOICE}\ \frac{\{p\}\ c_1\ \{q\} \qquad \{p\}\ c_2\ \{q\} \qquad b \text{ total}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}}$$

$$\textit{IFELSE}\ \frac{\{b^\# \wedge p\}\ c_1\ \{q\} \qquad \{(\neg b)^\# \wedge p\}\ c_2\ \{q\} \qquad b \text{ total and deterministic}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{q\}}$$

$$\textit{ASSERT}\ \frac{(\neg b)^\# \wedge q = \bot \qquad b \text{ deterministic}}{\{p +_b q\}\ \mathsf{assert}\ b^\#\ \{p\}} \qquad \textit{CONVEX}\ \frac{\{p_1\}\ c\ \{q_1\} \qquad \{p_2\}\ c\ \{q_2\} \qquad b \text{ constant}}{\{p_1 +_b p_2\}\ c\ \{q_1 +_b q_2\}}$$

$$\textit{MONOTONE}\ \frac{p_1 \leq p_2 \qquad \{p_2\}\ c\ \{q_2\} \qquad q_2 \leq q_1}{\{p_1\}\ c\ \{q_1\}} \qquad \textit{BOT}\ \frac{}{\{\bot\}\ c\ \{q\}}$$

Proof. The skip and comp rules follow from Theorem 29. The assign and sample rules follow from Theorem 24.

$$p = \mathsf{skip}\ ;p \qquad p \leq c_1\ ;q \leq c_1\ ;c_2\ ;r \qquad (u \coloneqq e)\ ;p = p[u \setminus e] \qquad (u \leftarrow s)\ ;p = p[u \setminus s]$$

The choice rule follows from *(i)* Theorem 98, *(ii)* the definition of command composition (Theorem 27), and *(iii)* the assumption.

$$\begin{aligned} & p && \overset{(i)}{=} \\ & (\mathsf{if}\ b\ \mathsf{then\ skip\ else\ skip})\ ;p && \overset{(ii)}{=} \\ & p +_b p && \overset{(iii)}{\leq} \\ & (c_1\ ;q) +_b (c_2\ ;q) && \overset{(ii)}{=} \\ & (\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2)\ ;q \end{aligned}$$

The ifelse rule is proven similarly, additionally using *(iv)* determinism of the guard b (Theorem 97) and *(v)* Theorem 29.

$$\begin{aligned} & p && \overset{(i)}{=} \\ & (\mathsf{if}\ b\ \mathsf{then\ skip\ else\ skip})\ ;p && \overset{(iv)}{=} \\ & (\mathsf{if}\ b\ \mathsf{then\ assert}\ b^\#\ \mathsf{else\ assert}\ (\neg b)^\#)\ ;p && \overset{(ii)}{=} \\ & (\mathsf{assert}\ b^\#\ ;p) +_b (\mathsf{assert}\ (\neg b)^\#\ ;p) && \overset{(v)}{=} \\ & (\mathsf{assert}(b^\# \wedge p)) +_b (\mathsf{assert}((\neg b)^\# \wedge p)) && \overset{(iii)}{\leq} \\ & (c_1\ ;q) +_b (c_2\ ;q) && \overset{(ii)}{=} \\ & (\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2)\ ;q \end{aligned}$$

The unroll rule applies Theorem 29.

$$p \leq \mathsf{if}\ b\ \mathsf{then}(c\ ;\mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else\ skip}\ ;q = \mathsf{while}\ b\ \mathsf{do}\ c\ ;q$$

The ASSERT rule follows from the definition of predicate combinators (Theorem 24), the assumption and determinism of b.

$$\text{assert}\, b^{\#} \,;\, p = p +_b \bot = p +_b (\neg b)^{\#} \wedge q = p +_b q$$

The CONVEX rule uses that constant guards commute with commands (Theorem 100).

$$p_1 +_b p_2 \leq (c\,;q_1) +_b (c\,;q_2) = c\,;(q_1 +_b q_2)$$

The MONOTONE rule uses monotonicity of composition. The BOT rule use the extra hypothesis.

$$p_1 \leq p_2 \leq c\,;q_2 \leq c\,;q_1 \qquad\qquad \bot \leq c\,;q$$

□

E Proofs for Section 6 (Distributive relational program logics)

We study the algebra of couplings.

Lemma 113. *Consider morphisms $c_i \colon X_i \to Y_i$ and $d_i \colon Y_i \to Z_i$ for $i = 1, 2$ in a commutative imperative category. If there are couplings $g \triangleright c_1 \,\&\, c_2$ and $h \triangleright d_1 \,\&\, d_2$, then there is a coupling $(g \mathbin{⨾} [h, d_1 \mathbin{⨾} \iota_1, d_2 \mathbin{⨾} \iota_2]) \triangleright (c_1 \mathbin{⨾} d_1) \,\&\, (c_2 \mathbin{⨾} d_2)$.*

Proof.

$$\begin{aligned}
&g \mathbin{⨾} [h, d_1 \mathbin{⨾} \iota_1, d_2 \mathbin{⨾} \iota_2] \mathbin{⨾} [\pi_1, \mathrm{id}, \mathbb{0}] && = \\
&g \mathbin{⨾} \iota \mathbin{⨾} [(h \mathbin{⨾} [\pi_1, \mathrm{id}, \mathbb{0}]), d_1] && = \\
&g \mathbin{⨾} \iota \mathbin{⨾} [(\pi \mathbin{⨾} d_1), d_1] && = \\
&g \mathbin{⨾} [\pi_1, \mathrm{id}, \mathbb{0}] \mathbin{⨾} d_1 && = \\
&\pi \mathbin{⨾} c_1 \mathbin{⨾} d_1
\end{aligned}$$

Similarly, one shows that $g \mathbin{⨾} [h, d_1 \mathbin{⨾} \iota_1, d_2 \mathbin{⨾} \iota_2] \mathbin{⨾} [\pi_2, \mathbb{0}, \mathrm{id}] = \pi \mathbin{⨾} c_2 \mathbin{⨾} d_2$. □

Lemma 114. *For two total morphisms $c_1 \colon X_1 \to Y_1$ and $c_2 \colon X_2 \to Y_2$ in a commutative imperative category, their monoidal product always gives a coupling: $((c_1 \otimes c_2) \mathbin{⨾} \iota_1) \triangleright c_1 \,\&\, c_2$.*

Proof. We use totality of c_2.

$$\begin{aligned}
&(c_1 \otimes c_2) \mathbin{⨾} \iota_1 \mathbin{⨾} [\pi_1, \mathrm{id}, \mathbb{0}] && = \\
&(c_1 \otimes c_2) \mathbin{⨾} \pi_1 && = \\
&\pi_1 \mathbin{⨾} c_1
\end{aligned}$$

Similarly, one shows that $(c_1 \otimes c_2) \mathbin{⨾} \iota_1 \mathbin{⨾} [\pi_2, \mathbb{0}, \mathrm{id}] = \pi \mathbin{⨾} c_2$ by totality of c_1. □

Lemma 115. *For two morphisms $c_1 \colon X_1 \to Y_1$ and $c_2 \colon X_2 \to Y_2$ in a commutative imperative category, a coupling $h \triangleright c_1 \,\&\, c_2$ induces a coupling $(\sigma \mathbin{⨾} h \mathbin{⨾} (\sigma + \sigma^{+})) \triangleright c_2 \,\&\, c_1$.*

Proof. This is easily checked as symmetries are isomorphisms. □

Lemma 116. *Consider morphisms $c_i, d_i \colon X_i \to Y_i$ and total morphisms $b_i \colon X_i \to 1 + 1$ for $i = 1, 2$ in a commutative imperative category. If there are couplings $g \triangleright c_1 \,\&\, c_2$, $g' \triangleright c_1 \,\&\, d_2$, $h' \triangleright d_1 \,\&\, c_2$, and $h \triangleright d_1 \,\&\, d_2$, then there is a coupling $l \triangleright (\text{if } b_1 \text{ then } c_1 \text{ else } d_1) \,\&\, (\text{if } b_2 \text{ then } c_2 \text{ else } d_2)$ defined by $l = \text{if } b_1 \text{ then}(\text{if } b_2 \text{ then } g \text{ else } g') \text{ else}(\text{if } b_2 \text{ then } h' \text{ else } h)$.*

Lemma 117. *We use that b_2 is total.*

$$
\begin{aligned}
& l \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}] && = \\
& (\mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}\ g\ \mathsf{else}\ g')\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}\ h'\ \mathsf{else}\ h)) \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}] && = \\
& \ \mathsf{if}\ b_1\ \mathsf{then}\ (\mathsf{if}\ b_2\ \mathsf{then}(g \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}])\ \mathsf{else}(g' \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}])) \\
& \qquad \mathsf{else}\ (\mathsf{if}\ b_2\ \mathsf{then}(h' \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}])\ \mathsf{else}(h \fatsemi [\pi_1, \mathrm{id}, \mathbb{0}])) && = \\
& \ \mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}(\pi_1 \fatsemi c_1)\ \mathsf{else}(\pi_1 \fatsemi c_1))\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}(\pi_1 \fatsemi d_1)\ \mathsf{else}(\pi_1 \fatsemi d_1)) && = \\
& \ \mathsf{if}\ b_1\ \mathsf{then}(\pi_1 \fatsemi c_1)\ \mathsf{else}(\pi_1 \fatsemi d_1) && = \\
& \pi_1 \fatsemi (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1)
\end{aligned}
$$

Similarly, one shows that $l \fatsemi [\pi_2, \mathbb{0}, \mathrm{id}] = \pi_2 \fatsemi (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)$ *using that b_1 is total.*

Lemma 118. *Consider morphisms $c_i, d_i \colon X_i \to Y_i$ and total and deterministic morphisms $b_i \colon X_i \to 1+1$ for $i = 1, 2$ in a commutative imperative category. If there are couplings $g \triangleright c_1 \mathbin{\&} c_2$ and $h \triangleright d_1 \mathbin{\&} d_2$, then there is a coupling $l \triangleright (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \mathbin{\&} (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)$ defined by $l = \mathsf{assert}(b_1 = b_2)\ ;(\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}\ g\ \mathsf{else}\ h)$.*

Proof sketch. The proof follows the same idea as that of Theorem 116, but additionally uses determinism of the guards to duplicate them in the assertion. □

Lemma 119. *Consider morphisms $c_i \colon X_i \to X_i$ and total and deterministic morphisms $b_i \colon X_i \to 1+1$ for $i = 1, 2$ in a commutative imperative category. If there is a coupling $g \triangleright c_1 \mathbin{\&} c_2$, then there is a coupling $l_d(g) \triangleright (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \mathbin{\&} (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)$ defined by*

$$
\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\{x'.\boldsymbol{\gamma}(x')\}\{y'.\boldsymbol{\delta}(y')\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}.
$$

Lemma 120. *Consider morphisms $c_i \colon X_i \to X_i$ and total morphisms $b_i \colon X_i \to 1+1$ for $i = 1, 2$ in a commutative imperative category. If there are couplings $g \triangleright c_1 \mathbin{\&} c_2$, $h_1 \triangleright c_1 \mathbin{\&} \mathrm{id}_{X_2}$, and $h_2 \triangleright \mathrm{id}_{X_1} \mathbin{\&} c_2$, then there is a coupling $l(g, h_1, h_2) \triangleright (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \mathbin{\&} (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)$.*

$$
\begin{aligned}
&\mathbf{loop}\ (\boldsymbol{\alpha}(x,y), \boldsymbol{\beta}_1(x_1,y_1), \boldsymbol{\beta}_2(x_2,y_2), \boldsymbol{\gamma}(x'), \boldsymbol{\delta}(y'))\{x,y,x_1,y_1,x_2,y_2,x',y'.(\\
&\quad (b_1 \otimes b_2)(x,y) \\
&\qquad \{g\{\boldsymbol{\alpha}(x,y)\}\{\boldsymbol{\delta}(x')\}\{\boldsymbol{\gamma}(y')\}\} \\
&\qquad \{h_1\{\boldsymbol{\beta}_1(x_1,y_1)\}\{\boldsymbol{\delta}(x')\}\{\boldsymbol{\gamma}'(y_o)\}\} \\
&\qquad \{h_2\{\boldsymbol{\beta}_2(x_2,y_2)\}\{\boldsymbol{\delta}'(x_o)\}\{\boldsymbol{\gamma}(y')\}\} \\
&\qquad \{\boldsymbol{\alpha}'(x_o,y_o)\} \\
&\quad + b_1(x_1,y_1)\{h_1\{\boldsymbol{\beta}_1(x_1,y_1)\}\{\boldsymbol{\delta}(x')\}\{\boldsymbol{\gamma}'(y_o)\}\}\{\boldsymbol{\alpha}'(x_o,y_o)\} \\
&\quad + b_2(x_2,y_2)\{h_2\{\boldsymbol{\beta}_2(x_2,y_2)\}\{\boldsymbol{\delta}'(x_o)\}\{\boldsymbol{\gamma}(y')\}\}\{\boldsymbol{\alpha}'(x_o,y_o)\} \\
&\quad + b_1(x')\{c_1\{\boldsymbol{\delta}(x')\}\}\{\boldsymbol{\delta}'(x_o)\} \\
&\quad + b_2(y')\{c_2\{\boldsymbol{\gamma}(y')\}\}\{\boldsymbol{\gamma}'(y_o)\})
\end{aligned}
$$

Theorem 88. *The following are valid relational assertion-correctness triples in any posetal imperative category where $\mathsf{abort} \le f$ for all morphisms f.*

$$
\textit{SKIP}\ \frac{}{\{p\}\ \mathsf{skip} \sim \mathsf{skip}\ \{p\}}
\qquad
\textit{COMP}\ \frac{\{p\}\ c_1 \sim d_1\ \{q\} \qquad \{q\}\ c_2 \sim d_2\ \{r\}}{\{p\}\ (c_1\ ;c_2) \sim (d_1\ ;d_2)\ \{r\}}
$$

$$
\textit{ASSIGN}\ \frac{e_1, e_2\ \textit{total and deterministic}}{\{p[(u_1,u_2) \setminus (e_1,e_2)]\}\ (u_1 \coloneqq e_1) \sim (u_2 \coloneqq e_2)\ \{p\}}
$$

$$\textsc{Choice}\ \frac{\{p\}\, c_1 \sim c_2\, \{q\} \qquad \{p\}\, c_1 \sim d_2\, \{q\} \qquad \{p\}\, d_1 \sim c_2\, \{q\} \qquad \{p\}\, d_1 \sim d_2\, \{q\} \qquad b_1, b_2 \ \mathit{total}}{\{p\}\, (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\, \{q\}}$$

$$\textsc{IfElse}\ \frac{\begin{array}{c}\{({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p\}\, c_1 \sim c_2\, \{q\} \\ \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}\, d_1 \sim d_2\, \{q\} \qquad b_1, b_2\ \mathit{total\ and\ deterministic}\end{array}}{\{(b_1 = b_2) \wedge p\}\, (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\, \{q\}}$$

$$\textsc{Loop}\ \frac{\{p\}\, c_1 \sim c_2\, \{p\} \qquad \{p\}\, c_1 \sim \mathsf{skip}\, \{p\} \qquad \{p\}\, \mathsf{skip} \sim c_2\, \{p\} \qquad b_1, b_2\ \mathit{total}}{\{p\}\, (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\, \{p\}}$$

$$\textsc{While}\ \frac{\{({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p\}\, c_1 \sim c_2\, \{(b_1 = b_2) \wedge p\} \qquad b_1, b_2\ \mathit{total\ and\ deterministic}}{\{(b_1 = b_2) \wedge p\}\, (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\, \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}}$$

$$\textsc{Monotone}\ \frac{p_1 \le p_2 \qquad \{p_2\}\, c \sim d\, \{q_2\} \qquad q_2 \le q_1}{\{p_1\}\, c \sim d\, \{q_1\}} \qquad \textsc{Symm}\ \frac{\{p\}\, c \sim d\, \{q\}}{\{\sigma \mathbin{;} p\}\, d \sim c\, \{\sigma \mathbin{;} q\}}$$

$$\textsc{Assign-L}\ \frac{e\ \mathit{total\ and\ deterministic}}{\{p[x \setminus e]\}\, (x \coloneqq e) \sim \mathsf{skip}\, \{p\}} \qquad \textsc{Choice-L}\ \frac{\{p\}\, c \sim \mathsf{skip}\, \{q\} \qquad \{p\}\, d \sim \mathsf{skip}\, \{q\} \qquad b\ \mathit{total}}{\{p\}\, (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d) \sim \mathsf{skip}\, \{q\}}$$

$$\textsc{IfElse-L}\ \frac{\{(b^{\#} \otimes \top) \wedge p\}\, c \sim \mathsf{skip}\, \{q\} \qquad \{((\neg b_1)^{\#} \otimes \top) \wedge p\}\, d \sim \mathsf{skip}\, \{q\} \qquad b\ \mathit{total\ and\ deterministic}}{\{p\}\, (\mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d) \sim \mathsf{skip}\, \{q\}}$$

$$\textsc{Loop-L}\ \frac{\{p\}\, c \sim \mathsf{skip}\, \{p\} \qquad b\ \mathit{total}}{\{p\}\, (\mathsf{while}\ b\ \mathsf{do}\ c) \sim \mathsf{skip}\, \{p\}} \qquad \textsc{While-L}\ \frac{\{(b^{\#} \otimes \top) \wedge p\}\, c \sim \mathsf{skip}\, \{p\} \qquad b\ \mathit{total\ and\ deterministic}}{\{p\}\, (\mathsf{while}\ b\ \mathsf{do}\ c) \sim \mathsf{skip}\, \{((\neg b)^{\#} \otimes \top) \wedge p\}}$$

Proof. skip. By Theorem 114, the monoidal product gives a coupling: $((\mathsf{skip} \otimes \mathsf{skip}) \fatsemi \iota) \triangleright \mathsf{skip} \mathbin{\&} \mathsf{skip}$. By unitality, we obtain the rule.

$$\mathsf{assert}\ p \mathbin{;} ((\mathsf{skip} \otimes \mathsf{skip}) \fatsemi \iota)^{=} = \mathsf{assert}\ p \mathbin{;} (\mathsf{skip} \otimes \mathsf{skip}) = \mathsf{assert}\ p$$

comp. Suppose there are couplings $g_1 \triangleright c_1 \& d_1$ and $g_2 \triangleright c_2 \& d_2$ satisfying $\mathsf{assert}\ p ; g_1^{=} \le g_1^{=} ; \mathsf{assert}\ q$ and $\mathsf{assert}\ q ; g_2^{=} \le g_2^{=} ; \mathsf{assert}\ r$. By Theorem 113, there is a coupling $(g_1 \fatsemi [g_2, c_2 \fatsemi \iota, d_2 \fatsemi \iota]) \triangleright (c_1 \fatsemi c_2) \& (d_1 \fatsemi d_2)$. Then, applying the definition of $(-)^{=}$ and the assumptions, we obtain the desired inequality.

$$\begin{array}{ll}
\mathsf{assert}\ p \mathbin{;} (g_1 \fatsemi [g_2, c_2 \fatsemi \iota, d_2 \fatsemi \iota])^{=} & = \\
\mathsf{assert}\ p \mathbin{;} g_1 \mathbin{;} \pi_1^{+} \mathbin{;} g_2 \mathbin{;} \pi_1^{+} & = \\
\mathsf{assert}\ p \mathbin{;} g_1^{=} \mathbin{;} g_2^{=} & \le \\
g_1^{=} \mathbin{;} \mathsf{assert}\ q \mathbin{;} g_2^{=} & \le \\
g_1^{=} \mathbin{;} g_2^{=} \mathbin{;} \mathsf{assert}\ r & = \\
g_1 \mathbin{;} \pi_1^{+} \mathbin{;} g_2 \mathbin{;} \pi_1^{+} \mathbin{;} \mathsf{assert}\ r & = \\
(g_1 \fatsemi [g_2, c_2 \fatsemi \iota, d_2 \fatsemi \iota])^{=} \mathbin{;} \mathsf{assert}\ r &
\end{array}$$

assign. By Theorem 114, the monoidal product gives a coupling: $(((u_1 \coloneqq e_1) \otimes (u_2 \coloneqq e_2)) \fatsemi \iota) \triangleright (u_1 \coloneqq e_1) \mathbin{\&} (u_2 \coloneqq e_2)$. This coupling satisfies the triple by determinism of e_1 and e_2.

$$\mathsf{assert}(p[(u_1, u_2) \setminus (e_1, e_2)]) \mathbin{;} (((u_1 \coloneqq e_1) \otimes (u_2 \coloneqq e_2)) \fatsemi \iota)^{=} \qquad =$$

$$
\begin{aligned}
&\mathsf{assert}(p[(u_1,u_2)\setminus(e_1,e_2)])\ ;\ ((u_1 \coloneqq e_1)\otimes(u_2 \coloneqq e_2)) &&=\\
&((u_1 \coloneqq e_1)\otimes(u_2 \coloneqq e_2))\ ;\ \mathsf{assert}\ p &&=\\
&(((u_1 \coloneqq e_1)\otimes(u_2 \coloneqq e_2))\ ⨾\ \iota)^{=}\ ;\ \mathsf{assert}\ p
\end{aligned}
$$

Choice. The assumption gives us couplings as in the hypotheses Theorem 116, so that we obtain a coupling $\mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}\ g\ \mathsf{else}\ g')\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}\ h'\ \mathsf{else}\ h)$ of $\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1$ and $\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2$. We show that it satisfies the triple.

$$
\begin{aligned}
&\mathsf{assert}\ p\ ;\ (\mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}\ g\ \mathsf{else}\ g')\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}\ h'\ \mathsf{else}\ h))^{=} &&=\\
&\mathsf{assert}\ p\ ;\ (\mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}\ g^{=}\ \mathsf{else}\ g'^{=})\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}\ h'^{=}\ \mathsf{else}\ h^{=})) &&=\\
&\mathsf{if}\ b_1\ \mathsf{then}\ (\mathsf{if}\ b_2\ \mathsf{then}(\mathsf{assert}\ p\ ;\ g^{=})\ \mathsf{else}(\mathsf{assert}\ p\ ;\ g'^{=}))\\
&\qquad \mathsf{else}\ (\mathsf{if}\ b_2\ \mathsf{then}(\mathsf{assert}\ p\ ;\ h'^{=})\ \mathsf{else}(\mathsf{assert}\ p\ ;\ h^{=})) &&\le\\
&\mathsf{if}\ b_1\ \mathsf{then}\ (\mathsf{if}\ b_2\ \mathsf{then}(g^{=}\ ;\ \mathsf{assert}\ q)\ \mathsf{else}(g'^{=}\ ;\ \mathsf{assert}\ q))\\
&\qquad \mathsf{else}\ (\mathsf{if}\ b_2\ \mathsf{then}(h'^{=}\ ;\ \mathsf{assert}\ q)\ \mathsf{else}(h^{=}\ ;\ \mathsf{assert}\ q)) &&=\\
&(\mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}\ g^{=}\ \mathsf{else}\ g'^{=})\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}\ h'^{=}\ \mathsf{else}\ h^{=}))\ ;\ \mathsf{assert}\ q &&=\\
&(\mathsf{if}\ b_1\ \mathsf{then}(\mathsf{if}\ b_2\ \mathsf{then}\ g\ \mathsf{else}\ g')\ \mathsf{else}(\mathsf{if}\ b_2\ \mathsf{then}\ h'\ \mathsf{else}\ h))^{=}\ ;\ \mathsf{assert}\ q
\end{aligned}
$$

Ifelse. The assumptions give us couplings as in the hypotheses of Theorem 118, so we obtain that $\mathsf{assert}(b_1 = b_2)\,;(\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}\ g\ \mathsf{else}\ h)$ is a coupling of $\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1$ and $\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2$. Then, we derive the inequality using determinism of b_1 and b_2, the definition of $(-)^{=}$, and the assumption.

$$
\begin{aligned}
&\mathsf{assert}(b_1 = b_2)\ ;\ \mathsf{assert}\ p\ ;\ (\mathsf{assert}(b_1 = b_2)\ ;\ (\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}\ g\ \mathsf{else}\ h))^{=} &&=\\
&\mathsf{assert}(b_1 = b_2)\ ;\ \mathsf{assert}\ p\ ;\ (\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}\ g^{=}\ \mathsf{else}\ h^{=}) &&=\\
&\mathsf{assert}(b_1 = b_2)\ ;\ (\mathsf{if}\ b_1\ \mathsf{then}\ (\mathsf{assert}({b_1}^{\#} \otimes {b_2}^{\#})\ ;\ \mathsf{assert}\ p\ ;\ g^{=})\\
&\qquad \mathsf{else}\ (\mathsf{assert}((\neg b_1)^{\#} \otimes (\neg b_2)^{\#})\ ;\ \mathsf{assert}\ p\ ;\ h^{=})) &&\le\\
&\mathsf{assert}(b_1 = b_2)\ ;\ (\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}(g^{=}\ ;\ \mathsf{assert}\ q)\ \mathsf{else}(h^{=}\ ;\ \mathsf{assert}\ q)) &&=\\
&\mathsf{assert}(b_1 = b_2)\ ;\ (\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}\ g^{=}\ \mathsf{else}\ h^{=})\ ;\ \mathsf{assert}\ q &&=\\
&(\mathsf{assert}(b_1 = b_2)\ ;\ (\mathsf{if}(b_1 \otimes b_2)\ \mathsf{then}\ g\ \mathsf{else}\ h))^{=}\ ;\ \mathsf{assert}\ q
\end{aligned}
$$

While. We use the assumption, determinism of b_1 and b_2, and Theorem 119.

$$
\begin{aligned}
&\mathsf{assert}(p \wedge (b_1 = b_2))\ ;\ (b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}) &&=\\
&b_1(x)\{b_2(y)\{(\mathsf{assert}(p \wedge ({b_1}^{\#} \otimes {b_2}^{\#}))\ ;\ g^{=})(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\}\\
&\qquad \{x,y.\ \mathsf{assert}(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\}\}\\
&\qquad \{x,y.\ \mathsf{assert}(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\} &&\le\\
&b_1(x)\{b_2(y)\{(g^{=}\ ;\ \mathsf{assert}\ p)(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\}\\
&\qquad \{x,y.\ \mathsf{assert}(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\}\}\\
&\qquad \{x,y.\ \mathsf{assert}(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\}
\end{aligned}
$$

Then, by uniformity, we obtain the desired inequality.

$$
\begin{aligned}
&\mathsf{assert}(p \wedge (b_1 = b_2))\ ;\ (\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g(x,y)\\
&\quad \{x,y.\boldsymbol{\alpha}(x,y)\}\{x'.\boldsymbol{\gamma}(x')\}\{y'.\boldsymbol{\delta}(y')\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\})^{=} &&=\\
&\mathsf{assert}(p \wedge (b_1 = b_2))
\end{aligned}
$$

$$
\begin{aligned}
&\quad ; (\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}) && \leq \\
&\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\} \\
&\quad \{x,y.\ \mathsf{assert}(p \wedge (\neg b_1{}^{\#} \otimes \neg b_2{}^{\#})p)\boldsymbol{\beta}(x,y)\}\}\{x,y.\ \mathsf{assert}(p \wedge (\neg b_1{}^{\#} \otimes \neg b_2{}^{\#})p)\boldsymbol{\beta}(x,y)\}\} && = \\
&(\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}) \\
&\quad ; \mathsf{assert}(p \wedge (\neg b_1{}^{\#} \otimes \neg b_2{}^{\#}))
\end{aligned}
$$

The derivation for the LOOP rule follows the same idea of that for the WHILE rule: it relies on Theorem 120 and uniformity, but it does not need determinism of the guards because they don't need to be duplicated in the pre- and post-conditions.

MONOTONE. Let $h \triangleright c \;\&\; d$ be the coupling given by the assumption.

$$
\mathsf{assert}\ p_1 \,;\, h^{=} \leq \mathsf{assert}\ p_2 \,;\, h^{=} \leq h^{=} \,;\, \mathsf{assert}\ q_2 \leq h^{=} \,;\, \mathsf{assert}\ q_1
$$

SYMM. Let $h \triangleright c \& d$ be the coupling given by the assumption. By Theorem 115, $(\sigma ⨟ h ⨟ (\sigma + \sigma^{+})) \triangleright d \& c$ and this satisfies the desired inequality.

$$
\begin{aligned}
&\mathsf{assert}(\sigma \,;\, p) \,;\, (\sigma ⨟ h ⨟ (\sigma + \sigma^{+}))^{=} && = \\
&\mathsf{assert}(\sigma \,;\, p) \,;\, \sigma \,;\, h^{=} \,;\, \sigma && = \\
&\sigma \,;\, \mathsf{assert}\ p \,;\, h^{=} \,;\, \sigma && \leq \\
&\sigma \,;\, h^{=} \,;\, \mathsf{assert}\ q \,;\, \sigma && = \\
&\sigma \,;\, h^{=} \,;\, \sigma \,;\, \mathsf{assert}(\sigma \,;\, q) && = \\
&(\sigma \,;\, h \,;\, (\sigma + \sigma^{+}))^{=} \,;\, \mathsf{assert}(\sigma \,;\, q)
\end{aligned}
$$

The one-sided rules are particular instances of the two sided rules, by taking (ASSIGN-L) the expression e_2 to be the variable u_2, (CHOICE-L, IFELSE-L) the commands c_2 and d_2 to be skip and (LOOP-L, WHILE-L) the guard b_2 to be $\mathbf{R}$ and the command c_2 to be skip. □

Theorem 90. *The following are valid relational predicate-incorrectness triples in any posetal imperative category where* abort $\leq f$ *and* $f ⨟ \top \leq \top$ *for all morphisms* f.

$$
\textit{SKIP}\ \frac{}{\{p\}\ \mathsf{skip} \sim \mathsf{skip}\ \{p\}}
\qquad
\textit{COMP}\ \frac{\{p\}\ c_1 \sim d_1\ \{q\} \qquad \{q\}\ c_2 \sim d_2\ \{r\}}{\{p\}\ (c_1 \,;\, c_2) \sim (d_1 \sim d_2)\ \{r\}}
$$

$$
\textit{ASSIGN}\ \frac{}{\{p[(u_1,u_2) \setminus (v_1,v_2)]\}\ (u_1 \coloneqq v_1) \sim (u_2 \coloneqq v_2)\ \{p\}}
\qquad
\textit{SAMPLE}\ \frac{h \triangleright c_1 \;\&\; c_2}{\{p[(u_1,u_2) \setminus h^{=}]\}\ (u_1 \leftarrow c_1) \sim (u_2 \leftarrow c_2)\ \{p\}}
$$

$$
\textit{CHOICE}\ \frac{\{p\}\ c_1 \sim c_2\ \{q\} \qquad \{p\}\ c_1 \sim d_2\ \{q\} \qquad \{p\}\ d_1 \sim c_2\ \{q\} \qquad \{p\}\ d_1 \sim d_2\ \{q\} \qquad b_1, b_2\ \textit{total}}{\{p\}\ (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\ \{q\}}
$$

$$
\textit{IFELSE}\ \frac{\begin{array}{c}\{(b_1{}^{\#} \otimes b_2{}^{\#}) \wedge p\}\ c_1 \sim c_2\ \{q\} \\ \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}\ d_1 \sim d_2\ \{q\} \qquad b_1, b_2\ \textit{total and deterministic}\end{array}}{\{(b_1 = b_2) \wedge p\}\ (\mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1) \sim (\mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2)\ \{q\}}
$$

$$
\textit{LOOP}\ \frac{\{p\}\ c_1 \sim c_2\ \{p\} \qquad \{p\}\ c_1 \sim \mathsf{skip}\ \{p\} \qquad \{p\}\ \mathsf{skip} \sim c_2\ \{p\} \qquad b_1, b_2\ \textit{total}}{\{p\}\ (\mathsf{while}\ b_1\ \mathsf{do}\ c_1) \sim (\mathsf{while}\ b_2\ \mathsf{do}\ c_2)\ \{p\}}
$$

$$\textsc{While}\quad \frac{\{({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p\}\; c_1 \sim c_2\; \{(b_1 = b_2) \wedge p\} \qquad b_1, b_2 \textit{ total and deterministic}}{\{(b_1 = b_2) \wedge p\}\; (\mathsf{while}\, b_1\, \mathsf{do}\, c_1) \sim (\mathsf{while}\, b_2\, \mathsf{do}\, c_2)\; \{((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p\}}$$

$$\textsc{Monotone}\quad \frac{p_1 \geq p_2 \qquad \{p_2\}\; c \sim d\; \{q_2\} \qquad q_2 \geq q_1}{\{p_1\}\; c \sim d\; \{q_1\}} \qquad \textsc{Choice-L}\quad \frac{\{p\}\; c \sim \mathsf{skip}\; \{q\} \qquad \{p\}\; d \sim \mathsf{skip}\; \{q\} \qquad b \textit{ total}}{\{p\}\; (\mathsf{if}\, b\, \mathsf{then}\, c\, \mathsf{else}\, d) \sim \mathsf{skip}\; \{q\}}$$

$$\textsc{Symm}\quad \frac{\{p\}\; c \sim d\; \{q\}}{\{\sigma\, ;\, p\}\; d \sim c\; \{\sigma\, ;\, q\}} \qquad \textsc{Assign-L}\quad \frac{}{\{p[x \setminus v]\}\; (x \coloneqq v) \sim \mathsf{skip}\; \{p\}} \qquad \textsc{Sample-L}\quad \frac{c \textit{ total}}{\{p[u \setminus c]\}\; (u \leftarrow c) \sim \mathsf{skip}\; \{p\}}$$

$$\textsc{IfElse-L}\quad \frac{\{(b^{\#} \otimes \top) \wedge p\}\; c \sim \mathsf{skip}\; \{q\} \qquad \{((\neg b_1)^{\#} \otimes \top) \wedge p\}\; d \sim \mathsf{skip}\; \{q\} \qquad b \textit{ total and deterministic}}{\{p\}\; (\mathsf{if}\, b\, \mathsf{then}\, c\, \mathsf{else}\, d) \sim \mathsf{skip}\; \{q\}}$$

$$\textsc{Loop-L}\quad \frac{\{p\}\; c \sim \mathsf{skip}\; \{p\} \qquad b \textit{ total}}{\{p\}\; (\mathsf{while}\, b\, \mathsf{do}\, c) \sim \mathsf{skip}\; \{p\}} \qquad \textsc{While-L}\quad \frac{\{(b^{\#} \otimes \top) \wedge p\}\; c \sim \mathsf{skip}\; \{p\} \qquad b \textit{ total and deterministic}}{\{p\}\; (\mathsf{while}\, b\, \mathsf{do}\, c) \sim \mathsf{skip}\; \{((\neg b)^{\#} \otimes \top) \wedge p\}}$$

Proof. skip. By Theorem 114, the monoidal product gives a coupling: $((\mathsf{skip} \otimes \mathsf{skip}) \fatsemi \iota) \triangleright \mathsf{skip} \,\&\, \mathsf{skip}$. By unitality, we obtain the rule.

$$p = (\mathsf{skip} \otimes \mathsf{skip})\, ;\, p = ((\mathsf{skip} \otimes \mathsf{skip}) \fatsemi \iota)^{=}\, ;\, p$$

comp. Suppose there are couplings $g_1 \triangleright c_1 \,\&\, d_1$ and $g_2 \triangleright c_2 \,\&\, d_2$ satisfying $p \geq g_1^{=}\, ;\, q$ and $q \geq g_2^{=}\, ;\, r$. By Theorem 113, there is a coupling $(g_1 \fatsemi [g_2, c_2 \fatsemi \iota, d_2 \fatsemi \iota]) \triangleright (c_1 \fatsemi c_2) \,\&\, (d_1 \fatsemi d_2)$. Then, applying the definition of $(-)^{=}$ and the assumptions, we obtain the desired inequality.

$$\begin{aligned}
&(g_1 \fatsemi [g_2, c_2 \fatsemi \iota, d_2 \fatsemi \iota])^{=}\, ;\, r && = \\
&g_1\, ;\, \pi_1^{+}\, ;\, g_2\, ;\, \pi_1^{+}\, ;\, r && = \\
&g_1^{=}\, ;\, g_2^{=}\, ;\, r && \leq \\
&g_1^{=}\, ;\, q && \leq \\
&p
\end{aligned}$$

assign. By Theorem 114, the monoidal product gives a coupling: $(((u_1 \coloneqq e_1) \otimes (u_2 \coloneqq e_2)) \fatsemi \iota) \triangleright (u_1 \coloneqq e_1) \,\&\, (u_2 \coloneqq e_2)$. This coupling satisfies the triple by definition.

$$p[(u_1, u_2) \setminus (e_1, e_2)] = ((u_1 \coloneqq e_1) \otimes (u_2 \coloneqq e_2))\, ;\, p = ((u_1 \coloneqq e_1) \otimes (u_2 \coloneqq e_2) \fatsemi \iota)^{=}\, ;\, p$$

sample. Given a coupling $h \triangleright c_1 \,\&\, c_2$, the triple is satisfied by definition.

$$p[(u_1, u_2) \setminus h^{=}] = h^{=}\, ;\, p$$

choice. The assumption gives us couplings as in the hypotheses Theorem 116, so that we obtain a coupling $\mathsf{if}\, b_1\, \mathsf{then}(\mathsf{if}\, b_2\, \mathsf{then}\, g\, \mathsf{else}\, g')\, \mathsf{else}(\mathsf{if}\, b_2\, \mathsf{then}\, h'\, \mathsf{else}\, h)$ of $\mathsf{if}\, b_1\, \mathsf{then}\, c_1\, \mathsf{else}\, d_1$ and $\mathsf{if}\, b_2\, \mathsf{then}\, c_2\, \mathsf{else}\, d_2$. We show that it satisfies the triple using totality of the guards.

$$\begin{aligned}
&(\mathsf{if}\, b_1\, \mathsf{then}(\mathsf{if}\, b_2\, \mathsf{then}\, g\, \mathsf{else}\, g')\, \mathsf{else}(\mathsf{if}\, b_2\, \mathsf{then}\, h'\, \mathsf{else}\, h))^{=}\, ;\, q && = \\
&(\mathsf{if}\, b_1\, \mathsf{then}(\mathsf{if}\, b_2\, \mathsf{then}\, g^{=}\, \mathsf{else}\, g'^{=})\, \mathsf{else}(\mathsf{if}\, b_2\, \mathsf{then}\, h'^{=}\, \mathsf{else}\, h^{=}))\, ;\, q && = \\
&\mathsf{if}\, b_1\, \mathsf{then}(\mathsf{if}\, b_2\, \mathsf{then}(g^{=}\, ;\, q)\, \mathsf{else}(g'^{=}\, ;\, q))\, \mathsf{else}(\mathsf{if}\, b_2\, \mathsf{then}(h'^{=}\, ;\, q)\, \mathsf{else}(h^{=}\, ;\, q)) && \leq \\
&\mathsf{if}\, b_1\, \mathsf{then}(\mathsf{if}\, b_2\, \mathsf{then}\, p\, \mathsf{else}\, p)\, \mathsf{else}(\mathsf{if}\, b_2\, \mathsf{then}\, p\, \mathsf{else}\, p) && = \\
&p
\end{aligned}$$

IFELSE. The assumptions give us couplings as in the hypotheses of Theorem 118, so we obtain that $\mathsf{assert}(b_1 = b_2) ; (\mathsf{if}(b_1 \otimes b_2) \mathsf{ then } g \mathsf{ else } h)$ is a coupling of $\mathsf{if}\ b_1 \mathsf{ then } c_1 \mathsf{ else } d_1$ and $\mathsf{if}\ b_2 \mathsf{ then } c_2 \mathsf{ else } d_2$. Then, we derive the inequality using determinism of b_1 and b_2, the definition of $(-)^{=}$, and the assumption.

$$\begin{aligned}
&(\mathsf{assert}(b_1 = b_2) ; (\mathsf{if}(b_1 \otimes b_2) \mathsf{ then } g \mathsf{ else } h))^{=} ; q && = \\
&\mathsf{assert}(b_1 = b_2) ; (\mathsf{if}(b_1 \otimes b_2) \mathsf{ then } g^{=} \mathsf{ else } h^{=}) ; q && = \\
&\mathsf{assert}(b_1 = b_2) ; (\mathsf{if}(b_1 \otimes b_2) \mathsf{ then} (g^{=} ; q) \mathsf{ else} (h^{=} ; q)) && \leq \\
&\mathsf{assert}(b_1 = b_2) ; (\mathsf{if}(b_1 \otimes b_2) \mathsf{ then } ({b_1}^{\#} \otimes {b_2}^{\#}) \wedge p \mathsf{ else } ((\neg b_1)^{\#} \otimes (\neg b_2)^{\#}) \wedge p) && = \\
&\mathsf{assert}(b_1 = b_2) ; (\mathsf{if}(b_1 \otimes b_2) \mathsf{ then } p \mathsf{ else } p) && = \\
&\mathsf{assert}(b_1 = b_2) ; p = \\
&(b_1 = b_2) \wedge p
\end{aligned}$$

WHILE. We use the assumption, determinism of b_1 and b_2, and Theorem 119.

$$\begin{aligned}
&b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.((b_1 = b_2) \wedge p)\{\boldsymbol{\alpha}()\}\}\} \\
&\{x,y.(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\}\}\{x,y.(p \wedge ((\neg b_1)^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\} && \leq \\
&b_1(x)\{b_2(y)\{x,y.(p \wedge ({b_1}^{\#} \otimes {b_2}^{\#}))\{\boldsymbol{\alpha}()\}\}\} \\
&\{x,y.(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\}\}\{x,y.(p \wedge (\neg {b_1}^{\#} \otimes \neg {b_2}^{\#}))\{\boldsymbol{\beta}()\}\} && = \\
&\mathsf{assert}(p \wedge (b_1 = b_2)) ; (b_1(x)\{b_2(y)\{x,y.\boldsymbol{\alpha}()\}\}\{x,y.\boldsymbol{\beta}()\}\}\{x,y.\boldsymbol{\beta}()\}) && =
\end{aligned}$$

Then, by uniformity and the extra hypothesis, we obtain the desired inequality.

$$\begin{aligned}
&(p \wedge (b_1 = b_2)) && \geq \\
&\mathsf{assert}(p \wedge (b_1 = b_2)) ; (\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{x,y.\boldsymbol{\alpha}(x,y)\}\{x,y.\boldsymbol{\beta}()\}\}\{x,y.\boldsymbol{\beta}()\}\}) && \geq \\
&\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\} \\
&\quad \{x,y.(p \wedge ((\neg b_1)^{\#} \otimes \neg {b_2}^{\#}))\boldsymbol{\beta}()\}\}\{x,y.(p \wedge ((\neg b_1)^{\#} \otimes \neg {b_2}^{\#}))\boldsymbol{\beta}()\}\} && = \\
&(\mathbf{loop}\ \boldsymbol{\alpha}(x,y)\{b_1(x)\{b_2(y)\{g^{=}(x,y)\{x,y.\boldsymbol{\alpha}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}\{x,y.\boldsymbol{\beta}(x,y)\}\}) \\
&\quad ; (p \wedge ((\neg b_1)^{\#} \otimes \neg {b_2}^{\#}))
\end{aligned}$$

The derivation for the LOOP rule follows the same idea of that for the WHILE rule: it relies on Theorem 120 and uniformity, but it does not need determinism of the guards because they don't need to be duplicated in the pre- and post-conditions.

MONOTONE. Let $h \triangleright c \,\&\, d$ be the coupling given by the assumption.

$$p_1 \geq p_2 \geq h^{=} ; q_2 \geq h^{=} ; q_1$$

SYMM. Let $h \triangleright c \,\&\, d$ be the coupling given by the assumption. By Theorem 115, $(\sigma \mathbin{⨾} h \mathbin{⨾} (\sigma + \sigma^{+})) \triangleright d \,\&\, c$ and this satisfies the desired inequality.

$$\begin{aligned}
&(\sigma \mathbin{⨾} h \mathbin{⨾} (\sigma + \sigma^{+}))^{=} ; \sigma ; q && = \\
&\sigma ; h^{=} ; \sigma ; \sigma ; q && = \\
&\sigma ; h^{=} ; q && \leq \\
&\sigma ; p
\end{aligned}$$

The one-sided rules are particular instances of the two sided rules, by taking (ASSIGN-L) the expression e_2 to be the variable u_2, (SAMPLE-L) the command c to be skip, (CHOICE-L, IFELSE-L) the commands c_2 and d_2 to be skip and (LOOP-L, WHILE-L) the guard b_2 to be **R** and the command c_2 to be skip. □