# Bialgebraic Reasoning on Stateful Languages

SERGEY GONCHAROV, University of Birmingham, UK
STEFAN MILIUS, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
LUTZ SCHRÖDER, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
STELIOS TSAMPAS, University of Southern Denmark (SDU), Denmark
HENNING URBAT, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Reasoning about program equivalence in imperative languages is notoriously challenging, as the presence of states (in the form of variable stores) fundamentally increases the observational power of program terms. The key desideratum for any notion of equivalence is *compositionality*, guaranteeing that subprograms can be safely replaced by equivalent subprograms regardless of the context. To facilitate compositionality proofs and avoid boilerplate work, one would hope to employ the abstract bialgebraic methods provided by Turi and Plotkin's powerful theory of *mathematical operational semantics* (a.k.a. *abstract GSOS*) or its recent extension by Goncharov et al. to higher-order languages. However, multiple attempts to apply abstract GSOS to stateful languages have thus failed. We propose a novel approach to the operational semantics of stateful languages based on the formal distinction between *readers* (terms that expect an initial input store before being executed), and *writers* (running terms that have already been provided with a store). In contrast to earlier work, this style of semantics is fully compatible with abstract GSOS, and we can thus leverage the existing theory to obtain coinductive reasoning techniques. We demonstrate that our approach generates non-trivial compositionality results for stateful languages with first-order and higher-order store and that it flexibly applies to program equivalences at different levels of granularity, such as trace, cost, and natural equivalence.

CCS Concepts: • **Theory of computation → Categorical semantics**; **Operational semantics**.

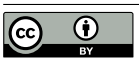
Additional Key Words and Phrases: Abstract GSOS, Imperative languages, Stateful languages

**ACM Reference Format:**
Sergey Goncharov, Stefan Milius, Lutz Schröder, Stelios Tsampas, and Henning Urbat. 2025. Bialgebraic Reasoning on Stateful Languages. *Proc. ACM Program. Lang.* 9, ICFP, Article 244 (August 2025), 30 pages. https://doi.org/10.1145/3747513

## 1 Introduction

Reasoning about program equivalence in stateful languages is a long-standing topic of interest in formal semantics with a sizeable body of work dedicated to it, e.g. [4–7, 9, 10, 18, 24, 25, 29, 31, 32, 34, 37–39, 42, 45, 48]. The level of attention the problem has received is due on the one hand to the fact that almost every popular programming language has some features involving mutable state, and on the other hand to the technical challenges posed by the design of formal reasoning methods for stateful languages, in particular in presence of higher-order store (e.g. [1, 40]). Efficient reasoning about program equivalence is largely a question of *compositionality*, which allows exchanging subprograms of a given program for equivalent ones without affecting the overall meaning of the

Authors' Contact Information: Sergey Goncharov, School of Computer Science, University of Birmingham, Birmingham, UK, s.goncharov@bham.ac.uk; Stefan Milius, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, stefan.milius@fau.de; Lutz Schröder, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, lutz.schroeder@fau.de; Stelios Tsampas, University of Southern Denmark (SDU), Odense, Denmark, stelios@imada.sdu.dk; Henning Urbat, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany, henning.urbat@fau.de.




ACM 2475-1421/2025/8-ART244
https://doi.org/10.1145/3747513

whole program. The well-known framework of *mathematical operational semantics* by Turi and Plotkin [41], also known as *abstract GSOS*, derives a generic setup for the operational semantics of programming and process languages from a form of categorical distributive law of syntax over behaviour (dubbed a *GSOS law*). It produces in particular a syntax-based *operational* model and a more abstract *denotational* model, and guarantees compositionality of the syntax with respect to the denotational semantics, thereby obviating the need for laborious ad-hoc compositionality proofs. Consequently, one would hope for abstract GSOS to provide a general and principled approach to establishing compositionality results for stateful languages.

However, there are known problems with applying abstract GSOS to a stateful setting: A naive modelling of stateful languages in the framework yields a denotational semantics in a final coalgebra for a functor involving the state monad. The character of such a semantics is that of a *resumption semantics*, which assumes arbitrary interference by the environment between program steps and as such is too fine-grained for most purposes (cf. Section 2.5). There have been two major attempts to address this issue. Abou-Saleh and Pattinson [2, 3] consider abstract GSOS laws over Kleisli categories for effect monads (such as the state monad). While the format of GSOS law is the same as in the original framework, the denotational semantics and the associated compositionality result need to be completely reworked for the purpose, and need complex technical assumptions. The coarsest form of semantics covered in their framework is *cost semantics*, which counts steps until termination. Contrastingly, the framework of *stateful SOS* introduced by Goncharov et al. [15] does also cover *trace semantics*, which considers the sequence of states computed by a program, and most notably *termination semantics* (also known as *natural semantics*), the standard end-to-end net execution semantics that only records the eventual result of a program. However, stateful SOS deviates from abstract GSOS even in the underlying abstraction of the rule format, replacing GSOS laws with *stateful SOS laws*, thus again requiring a dedicated redevelopment of the framework to obtain compositionality results. Moreover, the compositionality proofs in *op. cit.* are not inherently categorical and are based on syntactic reasoning at the level of rule formats.

In the present work, we propose a refined perspective on stateful languages that fully reconciles stateful operational semantics with abstract GSOS. The key novelty of our approach lies in an explicit division of program terms into *readers* (programs that expect an initial store to start their execution) and *writers* (programs that have already been given a store). For example, a standard imperative while-language with variable store can be extended to a reader-writer language where every sequential composition $p\,;q$ is regarded as a reader term and comes accompanied with a writer term $[p]_s\,;q$ for every store $s$, with $[p]_s$ representing the program $p$ executed on $s$.

The reader-writer separation, along with the explicit representation of states in the writer syntax, enables the instantiation of abstract GSOS theory: We demonstrate that reader-writer languages can be modelled in abstract GSOS over two-sorted sets, obtaining various flavours of stateful semantics (resumption, trace, cost, and termination semantics) as instances. In fact, our approach fits smoothly into a recent generalization of abstract GSOS due to Urbat et al. [43] that allows coverage of weak equivalences such as weak bisimilarity using a laxification of the core concept of bialgebra by Bonchi et al. [12]. It subsumes, inter alia, the previous compositionality results for stateful SOS [15] in the sense that every language specification in stateful SOS can be transformed into a two-sorted GSOS law in a semantics-preserving way.

We use a standard while-language as our running example. In a further, more advanced case study, we adapt the idea of reader-writer semantics to the recently introduced framework of *higher-order mathematical operational semantics* [16, 43], which extends abstract GSOS to higher-order languages. We then apply this abstract framework to a first-order language with higher-order store, similar to languages considered by Reus and Streicher [36] and by Pierce [33], obtaining compositionality of higher-order termination semantics.

$$\frac{p, s \to p', s'}{p\,;q, s \to p'\,;q, s'} \qquad \frac{p, s \downarrow s'}{p\,;q, s \to q, s'} \qquad \frac{}{\mathsf{skip}, s \downarrow s}$$

$$\frac{\mathsf{eev}(e, s) = n}{x \coloneqq e, s \downarrow s[x \coloneqq n]} \qquad \frac{\mathsf{eev}(e, s) = 0}{\mathsf{while}\ e\ p, s \downarrow s} \qquad \frac{\mathsf{eev}(e, s) \neq 0}{\mathsf{while}\ e\ p, s \to p\,;\mathsf{while}\ e\ p, s}$$

Fig. 1. Small-step operational semantics of Imp.

## 2 Bialgebraic Modelling of Stateful Languages

In this preliminary section we review the bialgebraic methods that the abstract GSOS framework [41] provides for reasoning about compositionality properties of programming languages. Moreover, we illustrate that stateful languages do not fit well into the framework when modelled naively, which motivates the refined approach presented subsequently in Section 3. As our running example we consider a simple imperative language, called Imp [47], with integer variables, assignments, sequential composition, and while loops.

### 2.1 The language Imp

We fix a countably infinite set $\mathcal{A}$ of *(program) variables*, and a set Ex of arithmetic expressions that are formed using standard operations such as $+, -, *$, constants $n \in \mathbb{Z}$, and variables. A *variable store* (or *state*) is a map $s\colon \mathcal{A} \to \mathbb{Z}$ assigning integer values to variables; we write $s[x \coloneqq n]$ for the store that maps $x$ to $n$ and otherwise equals $s$. We denote by $\mathcal{S}$ the set of states, and by $\mathsf{eev}\colon \mathsf{Ex} \times \mathcal{S} \to \mathbb{Z}$ the *expression evaluation* map; for example, $\mathsf{eev}(x * (y + 3) - y, s) = s(x) * (s(y) + 3) - s(y)$.

The set P of program terms of Imp is given by the grammar

$$\mathsf{P} \ni p, q ::= \ \mathsf{skip} \mid x \coloneqq e \mid \mathsf{while}\ e\ p \mid p\,;q$$

where $x \in \mathcal{A}$ and $e \in \mathsf{Ex}$. The (small-step) operational semantics of Imp are given by the inductive rules in Figure 1. The rules specify transitions of the form

$$p, s \to p', s' \qquad \text{and} \qquad p, s \downarrow s' \qquad (p, p' \in \mathsf{P},\ s, s' \in \mathcal{S})$$

stating that when the program $p$ is executed on store $s$, it transforms $s$ into $s'$ and then behaves like $p'$ (first case) or terminates (second case). These transitions are deterministic, hence induce a map

$$\gamma\colon \mathsf{P} \to (\mathsf{P} \times \mathcal{S} + \mathcal{S})^{\mathcal{S}} \qquad \text{given by} \qquad \gamma(p)(s) = \begin{cases} (p', s') & \text{if } p, s \to p', s', \\ s' & \text{if } p, s \downarrow s'. \end{cases} \tag{1}$$

There are several natural notions of program behaviour and program equivalence that can be associated to the transition system (1). Specifically, we consider the following four notions, presented in decreasing order w.r.t. the power of an observer (environment) that executes programs:

(1) *Resumption semantics* corresponds to a low-level observer that has unrestricted access to the current program state and can both read and modify it at any stage of the computation. Under this semantics, two programs are equivalent if they are indistinguishable regardless of how the observer interferes. This idea is formally captured by *resumption bisimilarity*:

**Definition 2.1.** A *resumption bisimulation* is a relation $R \subseteq \mathsf{P} \times \mathsf{P}$ such that for $R(p, q)$ and $s \in \mathcal{S}$,

(a) if $p, s \to p', s'$ then $\exists q'.\, q, s \to q', s' \wedge R(p', q')$;

(b) if $p, s \downarrow s'$ then $q, s \downarrow s'$;

(c) if $q, s \to q', s'$ then $\exists p'.\, p, s \to p', s' \wedge R(p', q')$;

(d) if $q, s \downarrow s'$ then $p, s \downarrow s'$.

*Resumption bisimilarity* $\sim_{\mathsf{res}}$ is the greatest resumption bisimulation (given by the union of all resumption bisimulations). It forms an equivalence relation, with the corresponding quotient map

$$\mathsf{res}\colon \mathsf{P} \to \mathsf{P}/{\sim_{\mathsf{res}}}.$$

Resumption bisimilarity yields a very fine-grained notion of program equivalence. For instance,

$$p = (x \coloneqq 1\,; x \coloneqq 2) \qquad \text{and} \qquad q = (x \coloneqq 1\,; x \coloneqq x+1) \tag{2}$$

are not bisimilar: if the observer sets the program state to any state $s$ with $s(x) \neq 1$ after the first step, then after the second step the states become different, hence distinguishable.

(2) *Trace semantics* corresponds to an observer that can read but not modify intermediate states. Formally, let $\mathcal{S}^+$ and $\mathcal{S}^\omega$ denote the sets of non-empty finite sequences and infinite sequences of elements of $\mathcal{S}$, respectively, and put $\mathcal{S}^\infty = \mathcal{S}^+ \cup \mathcal{S}^\omega$. The *trace map* $\mathsf{trc}\colon \mathsf{P} \to (\mathcal{S}^\infty)^{\mathcal{S}}$ associates to $p \in \mathsf{P}$ and $s \in \mathcal{S}$ the sequence of states produced by running $p$ on the initial state $s$:

$$\mathsf{trc}(p)(s) = \begin{cases} (s_1, \dots, s_n, s') \in \mathcal{S}^+ & \text{if } \exists p_i\ (1 \le i \le n).\ p, s \to p_1, s_1 \to \cdots \to p_n, s_n \downarrow s', \\ (s_1, s_2, s_3, \cdots) \in \mathcal{S}^\omega & \text{if } \exists p_i\ (i \ge 1). \qquad p, s \to p_1, s_1 \to p_2, s_2 \to p_3, s_3 \to \cdots. \end{cases}$$

(3) *Cost semantics* corresponds to an observer which can observe the fact that a computation step has happened (hence count the total number of steps) but can only access the program state after termination. This is captured by the following map $\mathsf{cst}\colon \mathsf{P} \to (\mathbb{N} \times \mathcal{S} + 1)^{\mathcal{S}}$, where $1 = \{*\}$:

$$\mathsf{cst}(p)(s) = \begin{cases} (n, s') & \text{if } \mathsf{trc}(p)(s) = (s_1, \dots, s_n, s') \in \mathcal{S}^+, \\ * & \text{if } \mathsf{trc}(p)(s) \in \mathcal{S}^\omega. \end{cases}$$

(4) *Termination semantics*, a.k.a. *natural semantics*, corresponds to an observer that can only view the terminating state (if any) of a program, while having no access to the actual computation. This is the standard form of semantics for imperative languages [47], and is captured by $\mathsf{ter}\colon \mathsf{P} \to (\mathcal{S}+1)^{\mathcal{S}}$:

$$\mathsf{ter}(p)(s) = \begin{cases} s' & \text{if } \mathsf{trc}(p)(s) = (s_1, \dots, s_n, s') \in \mathcal{S}^+, \\ * & \text{if } \mathsf{trc}(p)(s) \in \mathcal{S}^\omega. \end{cases}$$

Two programs $p$ and $q$ are *trace equivalent* if $\mathsf{trc}(p) = \mathsf{trc}(q)$, *cost equivalent* if $\mathsf{cst}(p) = \mathsf{cst}(q)$ and *termination equivalent* if $\mathsf{ter}(p) = \mathsf{ter}(q)$. Note that

$$\text{resumption bisimilarity} \Rightarrow \text{trace equivalence} \Rightarrow \text{cost equivalence} \Rightarrow \text{termination equivalence}.$$

For example, $p$ and $q$ of (2) are trace equivalent, hence also cost and termination equivalent.

**Remark 2.2.** The definitions of res, trc, cst and ter and the ensuing notions of semantic equivalence generalize to arbitrary transition systems of type $\gamma\colon X \to (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$ where, as above, we put

$$p, s \to p', s' \quad \text{if} \quad \gamma(p)(s) = (p', s') \qquad \text{and} \qquad p, s \downarrow s' \quad \text{if} \quad \gamma(p)(s) = s'.$$

The four forms of semantics are *compositional*, that is, they are respected by all Imp-constructors:

**Theorem 2.3** (Compositionality of Imp)**.** *For all* $p, p', q, q' \in \mathsf{P}$, $e \in \mathsf{Ex}$ *and* $[\![-]\!] \in \{\mathsf{res}, \mathsf{trc}, \mathsf{cst}, \mathsf{ter}\}$,

$$[\![p]\!] = [\![p']\!] \wedge [\![q]\!] = [\![q']\!] \implies [\![p\,; q]\!] = [\![p'\,; q']\!] \quad \textit{and} \quad [\![p]\!] = [\![p']\!] \implies [\![\mathsf{while}\ e\ p]\!] = [\![\mathsf{while}\ e\ p']\!].$$

The importance of this theorem is that it enables modular, inductive reasoning about complex programs: compositionality guarantees that the observable behaviour of a program with respect to either of the different semantics is fully determined by the behaviour of its subterms. Standardly, compositionality results such as Theorem 2.3 are proved from scratch, using inductive or coinductive methods tailored to the features of the language at hand. This approach has several drawbacks: compositionality proofs tend to be tedious and error-prone, while at the same time containing

many largely generic and repetitive parts. Additionally, every extension or modification of the language requires a meticulous adaptation of the proof. Our goal is to derive compositionality of stateful languages in a more principled manner by employing categorical techniques.

### 2.2 Categorical Background

The bialgebraic approach to compositional operational semantics, outlined in detail in Sections 2.3 and 2.4, rests on three fundamental categorical abstractions:

(1) Model the set of programs of a language as an (initial) algebra for a suitable syntax functor.
(2) Model the operational model of a language as a coalgebra for a suitable behaviour functor.
(3) Model program equivalences via coalgebraic (bi)simulations, parametric in a relation lifting.

**Remark 2.4.** Relation lifting [26] can be thought of as a systematic approach to *relational reasoning* over algebraic structures. The precise notion is given ahead in the text.

In the following we review the necessary terminology from category theory. Readers should be familiar with basic notions such as functors, natural transformations, (co)limits, and monads [30].

*Notation.* Given objects $X_1, X_2$ in a category $\mathbb{C}$, we write $X_1 \times X_2$ for their product, and $\langle f_1, f_2 \rangle \colon Y \to X_1 \times X_2$ for the pairing of morphisms $f_i \colon Y \to X_i$, $i = 1, 2$. Dually, $X_1 + X_2$ denotes the coproduct, $[g_1, g_2] \colon X_1 + X_2 \to Y$ the copairing of $g_i \colon X_i \to Y$, $i = 1, 2$, and we put $\nabla = [\mathrm{id}_X, \mathrm{id}_X] \colon X + X \to X$.

*Relations.* A *relation* on an object $X$ of $\mathbb{C}$ is a subobject (represented by a monomorphism) $R \rightarrowtail X \times X$. We write $\mathsf{l}_R, \mathsf{r}_R \colon R \to X$ for the left and right projections, and usually drop the subscript. A *morphism* from $R \rightarrowtail X \times X$ to a relation $S \rightarrowtail Y \times Y$ is a morphism $f \colon X \to Y$ of $\mathbb{C}$ such that there exists a (necessarily unique) dashed morphism making the first diagram in (3) below commute. Relations $R, S \rightarrowtail X \times X$ are ordered by $R \leq S$ iff $\mathrm{id}_X$ is a morphism from $R$ of $S$. We let $\mathbf{Rel}(\mathbb{C})$ denote the category of relations and their morphisms.

$$\begin{array}{ccc} R & \dashrightarrow & S \\ \downarrow & & \downarrow \\ X \times X & \xrightarrow{f \times f} & Y \times Y \end{array} \qquad\qquad \begin{array}{ccc} \mathbf{Rel}(\mathbb{D}) & \xrightarrow{\bar{F}} & \mathbf{Rel}(\mathbb{C}) \\ \downarrow & & \downarrow \\ \mathbb{D} & \xrightarrow{F} & \mathbb{C} \end{array} \tag{3}$$

In our applications we consider relations in the product category $\mathbf{Set}^T$ for a set $T$. Its objects are $T$-sorted sets $X = (X_t)_{t \in T}$, and a morphism $f \colon X \to Y$ is a $T$-sorted map $(f_t \colon X_t \to Y_t)_{t \in T}$.

**Example 2.5.** A relation $R \rightarrowtail X \times X$ in $\mathbf{Set}^T$ is given by a family $(R_t \subseteq X_t \times X_t)_{t \in T}$. We write $\Delta_X \rightarrowtail X \times X$ for the *identity relation*, i.e. $(\Delta_X)_t = \{(x, x) \mid x \in X_t\}$, and $R \bullet S$ for the *composite* of two relations $R, S \rightarrowtail X \times X$, i.e. $(R \bullet S)_t = R_t \bullet S_t$ where the right-hand $\bullet$ denotes the usual left-to-right composition of relations in $\mathbf{Set}$. The *product* of $R \rightarrowtail X \times X$ and $S \rightarrowtail Y \times Y$ is the relation $R \times S \rightarrowtail (X \times Y) \times (X \times Y)$ where $(R \times S)_t((x, y), (x', y'))$ iff $R_t(x, x')$ and $S_t(y, y')$. The *power* $R^B \rightarrowtail X^B \times X^B$ for a set $B$ is defined analogously. The *coproduct* $R + S \rightarrowtail (X + Y) \times (X + Y)$ is the disjoint union of $R$ and $S$.

**Example 2.6.** For the case of $T = 1$, where 1 is the singleton set, thus $\mathbb{C} = \mathbf{Set}^1 \cong \mathbf{Set}$, one recovers the standard notion of a relation on sets. For the case of $T = 2$, where 2 is the two-element set, relations on $\mathbf{Set}^2$ are *pairs* of set-theoretic relations etc.

A *relation lifting* of a functor $F \colon \mathbb{D} \to \mathbb{C}$ is a functor $\bar{F}$ making the second diagram in (3) commute, where the vertical arrows are the respective forgetful functors given by $(R \rightarrowtail X \times X) \mapsto X$. Every endofunctor $F$ on $\mathbf{Set}^T$ has a *canonical* relation lifting, which sends a relation $R \rightarrowtail X \times X$ to the

relation $\bar{F}R \rightarrowtail FX \times FX$ obtained via the (surjective, injective)-factorization shown below:

$$FR \twoheadrightarrow \bar{F}R \rightarrowtail FX \times FX \qquad \langle Fl_R, Fr_R \rangle$$

**Example 2.7.** (1) Recall that a *$T$-sorted algebraic signature*, for a fixed set $T$ of *sorts*, consists of a set $\Sigma$ of *operation symbols* and a map $\mathrm{ar}\colon \Sigma \to T^\star \times T$ associating to every $\mathsf{f} \in \Sigma$ its *arity*. We write $\mathsf{f}\colon t_1 \times \cdots \times t_n \to t$ if $\mathrm{ar}(\mathsf{f}) = (t_1, \ldots, t_n, t)$, and $\mathsf{f}\colon t$ if $n = 0$ (then $\mathsf{f}$ is called a *constant*). In the case of a single-sorted signature ($T = 1$) the arity of $\mathsf{f}$ is determined by the number $n$ of its arguments. Every signature $\Sigma$ induces an endofunctor on the category $\mathbf{Set}^T$, denoted by the same letter $\Sigma$ and defined by $(\Sigma X)_t = \coprod_{\mathsf{f}\colon t_1 \cdots t_n \to t} \prod_{i=1}^n X_{t_i}$ for $X \in \mathbf{Set}^T$ and $t \in T$. Endofunctors of this form are called *polynomial*. The canonical relation lifting of $\Sigma$ maps a relation $R \rightarrowtail X \times X$ to the relation $\overline{\Sigma}R \rightarrowtail \Sigma X \times \Sigma X$ where $(\overline{\Sigma}R)_t$ contains all pairs $(\mathsf{f}(x_1, \ldots, x_n), \mathsf{f}(y_1, \ldots, y_n))$ such that $\mathsf{f}\colon t_1 \times \cdots \times t_n \to t$ is in $\Sigma$ and $R_{t_i}(x_i, y_i)$ for $i = 1, \ldots, n$.
(2) The canonical relation lifting of the power set functor $\mathcal{P}\colon \mathbf{Set} \to \mathbf{Set}$ maps a relation $R \subseteq X \times X$ to the *Egli-Milner relation* $\overline{\mathcal{P}}R \subseteq \mathcal{P}X \times \mathcal{P}X$ where $\overline{\mathcal{P}}R(A, B)$ iff

$$\text{(i) } \forall a \in A.\, \exists b \in B.\, R(a, b) \;\wedge\; \text{(ii) } \forall b \in B.\, \exists a \in A.\, R(a, b).$$

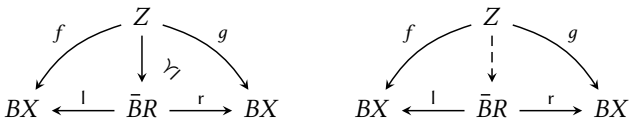
Requiring only (i) yields the non-canonical lifting $\vec{\mathcal{P}}$. Note that the relation $\vec{\mathcal{P}}R$ is *up-closed*, that is, $\vec{\mathcal{P}}R(A, B)$ and $B \subseteq B'$ implies $\vec{\mathcal{P}}R(A, B')$.

The up-closure of $\vec{\mathcal{P}}R$ turns out to be critical for the general compositionality result presented in Section 2.4. In abstract terms, this property is captured by imposing a suitable order structure:

**Definition 2.8** (Urbat et al. [43])**.** (1) A functor $B\colon \mathbb{D} \to \mathbb{C}$ is *ordered* if each hom-set $\mathbb{C}(Z, BX)$ ($Z \in \mathbb{C}, X \in \mathbb{D}$) is equipped with a preorder (a reflexive transitive relation) $\preceq$ such that

$$\forall q, q'\colon Z \to BX,\, p\colon Z' \to Z. \qquad q \preceq q' \implies q \cdot p \preceq q' \cdot p.$$

(2) A relation lifting $\bar{B}$ of an ordered functor $B\colon \mathbb{D} \to \mathbb{C}$ is *up-closed* if for every relation $R \rightarrowtail X \times X$ in $\mathbb{D}$, every span $BX \xleftarrow{f} Z \xrightarrow{g} BX$ in $\mathbb{C}$ and every morphism $Z \to \bar{B}R$ in $\mathbb{C}$ such that the left-hand triangle in the first diagram below commutes, and the right-hand triangle commutes laxly as indicated, there exists a morphism $Z \to \bar{B}R$ in $\mathbb{C}$ such that the second diagram commutes.




For instance, the lifting $\vec{\mathcal{P}}$ is up-closed if $\mathcal{P}$ is ordered by $q \preceq q'$ iff $\forall z \in Z.\, q(z) \subseteq q'(z)$.

*Algebras.* Given an endofunctor $\Sigma$ on a category $\mathbb{C}$, a *$\Sigma$-algebra* is a pair $(A, a)$ of an object $A$ and a morphism $a\colon \Sigma A \to A$. A *morphism* from $(A, a)$ to a $\Sigma$-algebra $(B, b)$ is a morphism $h\colon A \to B$ of $\mathbb{C}$ with $h \circ a = b \circ \Sigma h$. Algebras for $\Sigma$ and their morphisms form a category, and an *initial* $\Sigma$-algebra is simply an initial object in that category. We denote the initial algebra by $(\mu\Sigma, \iota)$ if it exists.

More generally, a *free $\Sigma$-algebra* on an object $X$ of $\mathbb{C}$ is a $\Sigma$-algebra $(\Sigma^\star X, \iota_X)$ together with a morphism $\eta_X\colon X \to \Sigma^\star X$ of $\mathbb{C}$ such that for every algebra $(A, a)$ and every morphism $h\colon X \to A$ of $\mathbb{C}$, there exists a unique $\Sigma$-algebra morphism $h^\star\colon (\Sigma^\star X, \iota_X) \to (A, a)$ such that $h = h^\star \circ \eta_X$. If $\mathbb{C}$ has an initial object 0, then $\Sigma^\star 0 = \mu\Sigma$. If free algebras exist on every object, then their formation induces a monad $\Sigma^\star\colon \mathbb{C} \to \mathbb{C}$, the *free monad* generated by $\Sigma$ [8]. Every $\Sigma$-algebra $(A, a)$ induces an Eilenberg-Moore algebra $\hat{a}\colon \Sigma^\star A \to A$, viz. $\hat{a} = \mathrm{id}_A^\star$ for the identity morphism $\mathrm{id}_A\colon A \to A$.

Given a relation lifting $\overline{\Sigma}$ of $\Sigma$, a $\overline{\Sigma}$-*congruence* on an algebra $(A, a)$ is a relation $R \rightarrowtail A \times A$ such that the algebra structure $a\colon \Sigma A \to A$ is a morphism $a\colon \overline{\Sigma} R \to R$ of $\mathbf{Rel}(\mathbb{C})$.

**Example 2.9.** The prototypical instance of the above categorical concepts are algebras for a signature. For every $T$-sorted signature $\Sigma$, an algebra for the associated polynomial functor $\Sigma$ is precisely an algebra for the signature $\Sigma$ in the usual sense, that is, an $T$-sorted set $A = (A_t)_{t\in T}$ equipped with an operation $\mathsf{f}^A\colon \prod_{i=1}^n A_{t_i} \to A_t$ for every $\mathsf{f}\colon t_1 \times \cdots \times t_n \to t$ in $\Sigma$. Morphisms of $\Sigma$-algebras are $T$-sorted maps respecting the algebraic structure. Given a $T$-sorted set $X$ of variables, the free algebra $\Sigma^\star X$ is the $\Sigma$-algebra of $\Sigma$-terms with variables from $X$; more precisely, $(\Sigma^\star X)_t$ is inductively defined by $X_t \subseteq (\Sigma^\star X)_t$ and $\mathsf{f}(u_1, \ldots, u_n) \in (\Sigma^\star X)_t$ for all $\mathsf{f}\colon t_1 \times \cdots \times t_n \to t$ and $u_i \in (\Sigma^\star X)_{t_i}$. The free algebra on the empty set is the initial algebra $\mu\Sigma$; it is formed by all *closed* $\Sigma$-terms. For every $\Sigma$-algebra $(A, a)$, the corresponding Eilenberg-Moore algebra $\hat{a}\colon \Sigma^\star A \to A$ is given by the map that evaluates terms in $A$.

For the canonical relation lifting $\overline{\Sigma}$, a $\overline{\Sigma}$-congruence on a $\Sigma$-algebra $A$ is the usual concept from universal algebra, namely a relation $R \rightarrowtail A \times A$ compatible with all operations: for $\mathsf{f}\colon t_1 \times \cdots \times t_n \to t$ and elements $x_i, y_i \in A_{t_i}$ such that $R_{t_i}(x_i, y_i)$ $(i = 1, \ldots, n)$, one has $R_t(\mathsf{f}^A(x_1, \ldots, x_n), \mathsf{f}^A(y_1, \ldots, y_n))$. Note that we do not require congruences to be equivalence relations.

**Example 2.10** (Program terms)**.** The set $\mathsf{P}$ of Imp-terms forms the initial algebra for the polynomial functor $\Sigma\colon \mathbf{Set} \to \mathbf{Set}$ corresponding the single-sorted signature of Imp:

$$\Sigma X = \underbrace{1}_{\mathsf{skip}} + \underbrace{\mathcal{A} \times \mathsf{Ex}}_{:=} + \underbrace{\mathsf{Ex} \times X}_{\mathsf{while}} + \underbrace{X \times X}_{-;-}. \tag{4}$$

Compositionality of Imp (Theorem 2.3) states that for $[\![-]\!] \in \{\mathsf{res}, \mathsf{trc}, \mathsf{cst}, \mathsf{ter}\}$ the equivalence relation $p \sim p' \iff [\![p]\!] = [\![p']\!]$ forms a congruence on the algebra $\mathsf{P} = \mu\Sigma$ of program terms.

*Coalgebras.* Dually to the notion of algebra, a *coalgebra* for an endofunctor $B$ on $\mathbb{C}$ is a pair $(C, c)$ of an object $C$ and a morphism $c\colon C \to BC$. A *morphism* from $(C, c)$ to a $B$-coalgebra $(D, d)$ is a morphism $h\colon C \to D$ of $\mathbb{C}$ such that $Bh \circ c = d \circ h$. Coalgebras for $B$ and their morphisms form a category, and a *final* $B$-coalgebra, denoted $(\nu B, \tau)$, is a final object in that category.

**Example 2.11.** Coalgebras form an abstraction of transition systems. For instance, the operational model $\gamma\colon \mathsf{P} \to (\mathsf{P} \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$ of Imp is a $D^{\mathcal{S}}$-coalgebra where $DX = X \times \mathcal{S} + \mathcal{S}$ on $\mathbf{Set}$.

The theory of coalgebras allows modelling notions of strong or weak (bi)simulation for transition systems at a convenient level of generality, using relation liftings of behaviour functors.

**Definition 2.12** (Urbat et al. [43])**.** Given a relation lifting $\bar{B}$ of $B$, a *weakening* [44] of a $B$-coalgebra $(C, c)$ is a $B$-coalgebra $(C, \tilde{c})$ such that for every relation $R \rightarrowtail C \times C$, there exists a morphism $R \to \bar{B}R$ in $\mathbb{C}$ making the first diagram below commute iff there exists a morphism $R \to \bar{B}R$ in $\mathbb{C}$ making the second diagram commute. A relation $R$ satisfying these two equivalent properties is then called a $\bar{B}$-*simulation on* $(C, c, \tilde{c})$. The greatest $\bar{B}$-simulation (if it exists) is called $\bar{B}$-*similarity*.

$$\begin{array}{ccc} R & \dashrightarrow & \bar{B}R \\ \downarrow & & \downarrow \\ C \times C & \xrightarrow{c \times \tilde{c}} & BC \times BC \end{array} \qquad\qquad \begin{array}{ccc} R & \dashrightarrow & \bar{B}R \\ \downarrow & & \downarrow \\ C \times C & \xrightarrow{\tilde{c} \times \tilde{c}} & BC \times BC \end{array} \tag{5}$$

**Remark 2.13.** Thanks to being parametric in the lifting $\bar{B}$ and the weakening $\tilde{c}$, the above notion of $\bar{B}$-simulation is quite flexible. In fact, despite the terminology, it will in some cases amount to *bi*simulations in the usual sense. There are two typical situations to which $\bar{B}$-simulations instantiate:

(1) For the trivial weakening $\tilde{c} = c$, the notion of $\bar{B}$-simulation first appeared in the work of Hermida and Jacobs [19] (under the name $\bar{B}$-*bi*simulation). For instance, for the power set functor $\mathcal{P}$ with its lifting $\vec{\mathcal{P}}$ of Example 2.7, a $\vec{\mathcal{P}}$-simulation is the usual notion of strong simulation of graphs, while the 'symmetric' lifting $\overline{\mathcal{P}}$ yields strong bisimulations.

(2) If $\tilde{c}$ is some form of reflexive transitive closure of $c$, then $\bar{B}$-simulations amount to a notion of *weak* simulation. For instance, let $c\colon C \to \mathcal{P}C$ and let $\tilde{c}\colon C \to \mathcal{P}C$ be given by $y \in \tilde{c}(x)$ iff there exist $n \geq 0$ and $x = x_0, \dots, x_n = y \in C$ such that $x_{i+1} \in c(x_i)$ for $0 \leq i < n$. Then a $\vec{\mathcal{P}}$-simulation on $(C, c, \tilde{c})$ corresponds to a weak simulation: given any two related states $x, y$, every strong transition from $x$ is matched by a weak transition from $y$. The fact that $\tilde{c}$ is a weakening expresses that this property is equivalent to matching weak transitions with weak transitions.

**Remark 2.14.** For endofunctors $B$ on $\mathbf{Set}^T$, we note that:

(1) The $\bar{B}$-similarity relation is given by the (sortwise) union of all $\bar{B}$-simulations on $(C, c, \tilde{c})$.

(2) If $\bar{B}$ is the canonical relation lifting, the final coalgebra $\nu B$ exists, the functor $B$ preserves weak pullbacks, and $\tilde{c} = c$, then two states of $(C, c)$ are $\bar{B}$-similar iff they are *behaviourally equivalent*, i.e. merged by the unique coalgebra morphism from $(C, c)$ to $\nu B$ [22, Thm. 4.2.4].

**Example 2.15.** The behaviour functor $D^{\mathcal{S}} = (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$ for Imp has a canonical lifting $\bar{D}^{\mathcal{S}}$ that sends $R \subseteq X \times X$ to $\bar{D}^{\mathcal{S}}R \subseteq (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}} \times (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$ where $\bar{D}^{\mathcal{S}}(f, g)$ iff, for all $s \in \mathcal{S}$, either $f(s) = (x, s')$ and $g(s) = (x', s')$ where $R(x, x')$ and $s' \in \mathcal{S}$, or $f(s) = g(s) \in \mathcal{S}$. Then $\bar{D}^{\mathcal{S}}$-similarity on the operational model (1) coincides with resumption bisimilarity (Definition 2.1). Since $D^{\mathcal{S}}$ preserves weak pullbacks, it also coincides with behavioural equivalence (Remark 2.14).

### 2.3 Abstract Operational Semantics

The abstract GSOS framework [41] yields an elegant categorical approach to operational semantics. It is parametric in two endofunctors $\Sigma, B\colon \mathbb{C} \to \mathbb{C}$ on a category $\mathbb{C}$ with products, where $\Sigma$ has an initial algebra $\mu\Sigma$ and generates a free monad $\Sigma^\star$. The functors $\Sigma$ and $B$ represent the *syntax* and *behaviour* of a programming language, with $\mu\Sigma$ thought of as the object of program terms. The operational semantics of a language are modelled by a *GSOS law of* $\Sigma$ *over* $B$: a natural transformation

$$\varrho_X\colon \Sigma(X \times BX) \to B\Sigma^\star X \quad (X \in \mathbb{C}). \tag{6}$$

Informally, $\varrho$ encodes the inductive operational rules of the language at hand: for every program constructor f, it specifies the one-step behaviour of programs $\mathsf{f}(p_1, \dots, p_n)$, i.e. the $\Sigma$-terms they transition into next, depending on the one-step behaviours of the operands $p_1, \dots, p_n$.

**Example 2.16.** The operational rules of Imp (Figure 1) can be translated into a GSOS law $\varrho$ of the signature functor $\Sigma$ given in Example 2.10 over the behaviour functor $D^{\mathcal{S}}X = (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$. The component $\varrho_X\colon \Sigma(X \times (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}) \to (\Sigma^\star X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$ encodes, for instance, the rules for sequential composition into the following assignment for $p, q \in X$ and $f, g \in (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$:

$$\varrho_X((p, f)\,;(q, g)) \;=\; \lambda s. \begin{cases} ((p'\,;q), s') & \text{if } f(s) = (p', s') \in X \times \mathcal{S}, \\ (q, s') & \text{if } f(s) = s' \in \mathcal{S}. \end{cases}$$

The universal property of the initial algebra $(\mu\Sigma, \iota)$ entails that there exists a unique $B$-coalgebra structure $\gamma\colon \mu\Sigma \to B(\mu\Sigma)$ such that the diagram below commutes:

$$\begin{array}{ccc} \Sigma(\mu\Sigma) & \xrightarrow{\iota} & \mu\Sigma \xrightarrow{\gamma} B(\mu\Sigma) \\ {\scriptstyle \Sigma\langle \mathrm{id}, \gamma\rangle}\downarrow & & \uparrow{\scriptstyle B\hat{\imath}} \\ \Sigma(\mu\Sigma \times B(\mu\Sigma)) & \xrightarrow{\varrho_{\mu\Sigma}} & B\Sigma^\star(\mu\Sigma) \end{array} \tag{7}$$

The coalgebra $(\mu\Sigma, \gamma)$ is called the *operational model* of the GSOS law $\varrho$. Informally, this is the transition system that runs programs according the operational rules represented by $\varrho$. For instance, the operational model of the GSOS law for Imp (Example 2.16) is precisely the transition system (1) on the set $\mathsf{P} = \mu\Sigma$ of program terms induced by the rules of Imp.

### 2.4 Compositionality in Abstract GSOS

The key feature of abstract GSOS is that it allows for general compositionality results applying uniformly to languages modelled in the framework. The compositionality theorem presented below (Theorem 2.20) is substantially more powerful than the original one by Turi and Plotkin [41]: it not only applies to behavioural equivalence in the final coalgebra, but to similarity w.r.t. a choice of relation lifting of the behaviour functor and a choice of weakening of the operational model. Let us first fix the required data. We restrict to the base category $\mathbb{C} = \mathbf{Set}^T$ and polynomial syntax functors, which simplifies some technical conditions and is sufficient to our purposes:

**Definition 2.17.** An *abstract operational setting* (*AOS*) $\mathcal{O} = (\Sigma, \overline{\Sigma}, B, \bar{B}, \varrho, \gamma, \tilde{\gamma})$ is given by the following data:

- a polynomial functor $\Sigma\colon \mathbf{Set}^T \to \mathbf{Set}^T$ with its canonical relation lifting $\overline{\Sigma}$;
- an ordered functor $(B, \preceq)\colon \mathbf{Set}^T \to \mathbf{Set}^T$ with a (not necessarily canonical) relation lifting $\bar{B}$;
- a GSOS law $\varrho$ of $\Sigma$ over $B$;
- the operational model $(\mu\Sigma, \gamma)$ of $\varrho$ with a weakening $\tilde{\gamma}$.

The point of the definition of AOS is to collect (in a neat, abstract form) the *problem setting* of relational reasoning about some "weak" notion of program equivalence on a (first-order) programming language. The polynomial functor $\Sigma$ models the algebraic signature of the language and $B$ models the behaviour, for instance $B = \mathcal{P}(L \times \mathrm{Id})$ for a nondeterministic LTS. The relation lifting $\overline{B}$ eventually determines the type of program equivalence considered. The GSOS law $\varrho$ corresponds to the collection of operational rules and the weakening $\overline{\gamma}$ to the chosen notion of a *weak* transition system, e.g. a saturation $\Rightarrow$.

Given an AOS we can study the congruence of $\bar{B}$-similarity on $(\mu\Sigma, \gamma, \tilde{\gamma})$. It turns out that the congruence property boils down to three natural conditions.

The first one is that the lifting $\bar{B}$ is up-closed and laxly preserves composition and identities:

$$\bar{B}R \bullet \bar{B}S \subseteq \bar{B}(R \bullet S) \text{ for all } R, S \rightarrowtail X \times X \qquad \text{and} \qquad \Delta_{BX} \subseteq \bar{B}\Delta_X \text{ for all } X,$$

where $\subseteq$ means sortwise inclusion. This ensures that $\bar{B}$-similarity is a preorder [43, Lemma VI.3].

The second condition is *liftability* of the law $\varrho$:

**Definition 2.18.** The GSOS law $\varrho$ of an AOS is *liftable* if for every relation $R \rightarrowtail X \times X$ in $\mathbf{Set}^T$, the component $\varrho_X$ is a $\mathbf{Rel}(\mathbf{Set}^T)$-morphism from $\overline{\Sigma}(R \times \bar{B}R)$ to $\bar{B}\,\overline{\Sigma}^*\,R$.

Note that the free monad $\overline{\Sigma}^*$ is given by the canonical lifting $\overline{\Sigma^*}$ of the free monad $\Sigma^*$ [43, Prop. V.4]. Intuitively, liftability expresses in abstract terms that the operational rules encoded by $\varrho$ are parametrically polymorphic, i.e. do not inspect the structure of their arguments. Hence,

every relation between the arguments should be preserved by $\varrho$. Liftability is closely related to dinaturality of $\varrho$; in fact, when using canonical liftings, dinaturality implies liftability (Remark 2.21).

The final condition ensures that weak transitions interact well with the operational rules:

**Definition 2.19.** The triple $(\mu\Sigma, \iota, \tilde{\gamma})$ is a *lax $\varrho$-bialgebra* if the diagram below commutes laxly:

$$\begin{array}{ccccc} \Sigma(\mu\Sigma) & \xrightarrow{\iota} & \mu\Sigma & \xrightarrow{\tilde{\gamma}} & B(\mu\Sigma) \\ {\scriptstyle \Sigma\langle \mathrm{id}, \tilde{\gamma}\rangle}\downarrow & \swarrow & & & \uparrow{\scriptstyle B\hat{\iota}} \\ \Sigma(\mu\Sigma \times B(\mu\Sigma)) & & \xrightarrow{\varrho_{\mu\Sigma}} & & B\Sigma^\star(\mu\Sigma) \end{array} \tag{8}$$

The lax bialgebra condition expresses that the operational rules corresponding to $\varrho$ remain sound in the operational model when strong transitions (given by $\gamma$) are replaced by weak transitions (given by $\tilde{\gamma}$). Lax bialgebras were introduced (in a slightly less general form than Definition 2.19) by Bonchi et al. [12] and originally used to analyse sound up-to techniques for weak (bi)similarity.

The following general compositionality result for abstract GSOS is a special case of [43, Cor. VIII.7]. A higher-order version will appear in Section 5.

**Theorem 2.20** (Compositionality)**.** *Suppose that $\mathcal{O} = (\Sigma, \overline{\Sigma}, B, \bar{B}, \varrho, \gamma, \tilde{\gamma})$ is an AOS such that*

(1) *$\bar{B}$ is up-closed and laxly preserves composition and identities;*

(2) *$\varrho$ is liftable;*

(3) *$(\mu\Sigma, \iota, \tilde{\gamma})$ is a lax $\varrho$-bialgebra.*

*Then the $\bar{B}$-similarity relation on $(\mu\Sigma, \gamma, \tilde{\gamma})$ is a $\overline{\Sigma}$-congruence on the initial algebra $(\mu\Sigma, \iota)$.*

**Remark 2.21.** In practice, some of the conditions of the theorem may come for free:

(1) holds if $\bar{B}$ is canonical, and $B$ is ordered by equality and preserves weak pullbacks [44, Prop. C.9].

(2) holds if $\bar{B}$ is the canonical lifting [44, Constr. D.5].

(3) holds if $B$ is ordered by equality and $\tilde{\gamma} = \gamma$; in this case, the lax bialgebra condition (8) is just (7).

For functors $B$ preserving weak pullbacks and having a final coalgebra, Theorem 2.20 thus specializes to the original congruence result by Turi and Plotkin [41]: for every GSOS law, behavioural equivalence in the final coalgebra is a congruence on the operational model.

The general compositionality theorem simplifies proofs of congruence results for programming languages by reducing their congruence properties to elementary and usually easily verifiable conditions on the lifting $\bar{B}$ of the behaviour functor and the rules of the language given by $\varrho$.

### 2.5 The Fundamental Challenge

Can we deduce the compositionality of Imp (Theorem 2.3) from the compositionality theorem for abstract GSOS? At this stage, only the case of resumption semantics – the most fine-grained and arguably least useful of the four notions of program semantics in Section 2.1 – is immediate:

**Example 2.22.** We apply Theorem 2.20 to the AOS $\mathcal{O} = (\Sigma, \overline{\Sigma}, D^{\mathcal{S}}, \bar{D}^{\mathcal{S}}, \varrho, \gamma, \gamma)$ where $\Sigma$ is the signature of Imp, $D^{\mathcal{S}}$ and $\bar{D}^{\mathcal{S}}$ are as in Example 2.15 (with $D^{\mathcal{S}}$ ordered by equality), and $\varrho$ is as in Example 2.16. Recall that $\bar{D}^{\mathcal{S}}$-similarity is resumption bisimilarity. By Remark 2.21, all conditions of Theorem 2.20 are satisfied, whence resumption bisimilarity is a congruence on the algebra $\mathsf{P} = \mu\Sigma$ of Imp-terms. In fact, since $B$ has a final coalgebra, this result already follows from Turi and Plotkin's original theory of congruence and does not require the full generality of Theorem 2.20.

To capture the more interesting parts of Theorem 2.3 in our abstract setting, we would need to come up with liftings $\bar{D}^{\mathcal{S}}$ and weakenings $\tilde{\gamma}$ such that the ensuing notions of $\bar{D}^{\mathcal{S}}$-similarity coincide

with trace, cost, and termination equivalence, respectively. This appears to be impossible under the present choice of behaviour functor $D^{\mathcal{S}}X = (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$; intuitively, the exponent $(-)^{\mathcal{S}}$ forces a resumption-like semantics. Our solution lies in switching to a *two-sorted* refinement of Imp.

## 3 Readers and Writers

Trace semantics in Imp interprets a term $p$ as the function mapping an input store $s$ to the (possibly infinite) trace of $(p, s)$. This suggests that program-store pairs $(p, s)$ are essentially treated as the states of a transition system, i.e. $(p, s) \xrightarrow{s'} (p', s')$. On the other hand, the operational semantics treats terms as *transformers* acting on an input store: for instance, in $p; q$, the subterm $q$ is thought of as an 'open' computation, which needs to be given an input $s$ to start. Afterwards, however, trace semantics treats the pair $(q, s)$ as a *running* computation.

The reader-writer approach to stateful semantics turns this implicit divide between "transformers" and "started transformers/computations" into explicit syntactic sorts, namely *readers* and *writers* respectively. Readers represent programs that need to be provided with an input state in order to run. Contrastingly, writers represent programs that are already running, producing a trace of states, and for instance arise by passing an input state to a reader. We illustrate this idea via a language called $\mathsf{Imp}^2$, which extends the language Imp.

### 3.1 The Language $\mathsf{Imp}^2$

We inherit the sets $\mathcal{A}$ (program variables), $\mathcal{S}$ (states) and Ex (arithmetic expressions) from Imp. The program terms of $\mathsf{Imp}^2$ are split into a set R of readers $p, q, \ldots$ and a set W of writers $c, \ldots$ and are defined by the grammar

$$\mathsf{R} \ni p, q ::= \mathsf{skip} \mid x := e \mid \mathsf{while}\ e\ p \mid p\,;q \qquad \text{and} \qquad \mathsf{W} \ni c ::= [p]_s \mid s.c \mid \mathrm{ret}_s \mid c\,;q \tag{9}$$

where $s \in \mathcal{S}$, $x \in \mathcal{A}$ and $e \in \mathsf{Ex}$. Equivalently, $(\mathsf{R}, \mathsf{W})$ is the initial algebra for the polynomial endofunctor $\Sigma\colon \mathbf{Set}^2 \to \mathbf{Set}^2$ corresponding to the two-sorted signature of $\mathsf{Imp}^2$, with sorts r and w:

$$\Sigma_{\mathsf{r}}X = \underbrace{1}_{\mathsf{skip}} + \underbrace{\mathcal{A} \times \mathsf{Ex}}_{:=} + \underbrace{\mathsf{Ex} \times X_{\mathsf{r}}}_{\mathsf{while}} + \underbrace{X_{\mathsf{r}} \times X_{\mathsf{r}}}_{-;-}, \quad \Sigma_{\mathsf{w}}X = \underbrace{X_{\mathsf{r}} \times \mathcal{S}}_{[-]_s} + \underbrace{\mathcal{S} \times X_{\mathsf{w}}}_{s.-} + \underbrace{\mathcal{S}}_{\mathrm{ret}_s} + \underbrace{X_{\mathsf{w}} \times X_{\mathsf{r}}}_{-;-}.$$

Note that readers are precisely Imp-terms (P = R). Additionally, there are four types of writers: For every reader $p$ and state $s$, the writer $[p]_s$ represents the encapsulated computation of $p$ at input $s$. This bears some resemblance with the *fine-grain call-by-value* paradigm [28] with its operator $[-]$ casting values as computations. The writer $s.c$ produces the state $s$ and then behaves like $c$. The writer $\mathrm{ret}_s$ terminates in the state $s$. Finally, the writer $c\,;q$ represents a sequential composition that currently evaluates its first argument. One can think of writers as programs with 'explicit' store, in analogy with explicit substitutions: the store is gradually moved towards the computation point, e.g. one may derive $[p\,;q]_s \to [p]_s\,;q$ or $\mathrm{ret}_s\,;q \xrightarrow{s} [q]_s, s$, where a labelled transition denotes 'store output', signaling to the outer world that a computation step in Imp has been completed.

Formally, the operational semantics of readers are specified by the rules in the top part of Figure 2. They inductively determine transitions of the form

$$p, s \to c \quad \text{where} \quad p \in \mathsf{R},\ c \in \mathsf{W}, s \in \mathcal{S}, \tag{10}$$

to be read as 'on input state $s$, reader $p$ continues as writer $c$'. The reader semantics induces a map

$$\gamma_0^{\mathsf{r}}\colon \mathsf{R} \to \mathsf{W}^{\mathcal{S}} \qquad \text{given by} \qquad \gamma_0^{\mathsf{r}}(p)(s) = c \qquad \text{if} \qquad p, s \to c. \tag{11}$$

$$\frac{}{p\,;q, s \to [p]_s\,;q} \qquad \frac{}{\mathsf{skip}, s \to \mathsf{ret}_s} \qquad \frac{\mathsf{eev}(e,s) = n}{x \coloneqq e, s \to \mathsf{ret}_{s[x:=n]}}$$

$$\frac{\mathsf{eev}(e,s) = 0}{\mathsf{while}\ e\ p, s \to \mathsf{ret}_s} \qquad \frac{\mathsf{eev}(e,s) \neq 0}{\mathsf{while}\ e\ p, s \to s.[p\,;\mathsf{while}\ e\ p]_s}$$

$$\frac{p, s \to c}{[p]_s \to c} \qquad \frac{}{\mathsf{ret}_s \downarrow s} \qquad \frac{}{s.c \xrightarrow{s} c} \qquad \frac{c \xrightarrow{s} d}{c\,;q \xrightarrow{s} d\,;q} \qquad \frac{c \to d}{c\,;q \to d\,;q} \qquad \frac{c \downarrow s'}{c\,;q \xrightarrow{s'} [q]_{s'}}$$

Fig. 2. Reader semantics of $\mathsf{Imp}^2$ (top) and writer semantics of $\mathsf{Imp}^2$ (bottom).

The operational semantics of the writers are given in the bottom part of Figure 2. These rules determine transitions of type

$$c \to d, \qquad c \xrightarrow{s} d, \qquad c \downarrow s, \qquad \text{where} \qquad c, d \in \mathsf{W},\ s \in \mathcal{S},$$

to be read as '$c$ progresses to $d$', '$c$ progresses to $d$ and generates the state $s$', and '$c$ terminates in the state $s$', respectively. The writer semantics thus yields a function

$$\gamma_0^{\mathsf{w}} \colon \mathsf{W} \to \mathsf{W} \times \mathcal{S} + \mathsf{W} + \mathcal{S} \quad \text{given by} \quad \gamma_0^{\mathsf{w}}(c) = (d, s)\ /\ d\ /\ s \quad \text{if} \quad c \xrightarrow{s} d\ /\ c \to d\ /\ c \downarrow s. \tag{12}$$

**Notation 3.1.** We denote objects of $\mathbf{Set}^2$ as a pairs $X = (X_{\mathsf{r}}, X_{\mathsf{w}})$, morphisms as pairs $f = (f^{\mathsf{r}}, f^{\mathsf{w}})$[1], and we write $F_{\mathsf{r}}$ and $F_{\mathsf{w}}$ for the two components of a functor $F \colon \mathbb{C} \to \mathbf{Set}^2$.

The two maps (11) and (12) combine into a coalgebra

$$\gamma_0 = (\gamma_0^{\mathsf{r}}, \gamma_0^{\mathsf{w}}) \colon (\mathsf{R}, \mathsf{W}) \to B_0(\mathsf{R}, \mathsf{W}) \tag{13}$$

for the endofunctor $B_0$ on $\mathbf{Set}^2$ defined by

$$(B_0)_{\mathsf{r}} X = X_{\mathsf{w}}^{\mathcal{S}} \qquad \text{and} \qquad (B_0)_{\mathsf{w}} X = X_{\mathsf{w}} \times \mathcal{S} + X_{\mathsf{w}} + \mathcal{S}. \tag{14}$$

The coalgebra (13) is the operational model of the GSOS law $\varrho_0$ of $\Sigma$ over $B_0$ that encodes the operational rules of Figure 2. The component $\varrho_{0,X} = (\varrho_0)_X$ at $X \in \mathbf{Set}^2$ is given by the maps

$$\varrho_{0,X}^{\mathsf{r}} \colon \Sigma_{\mathsf{r}}(X \times B_0 X) \to (\Sigma_{\mathsf{w}}^{\star} X)^{\mathcal{S}} \qquad \text{and} \qquad \varrho_{0,X}^{\mathsf{w}} \colon \Sigma_{\mathsf{w}}(X \times B_0 X) \to \Sigma_{\mathsf{w}}^{\star} X \times \mathcal{S} + \Sigma_{\mathsf{w}}^{\star} X + \mathcal{S}$$

where, for instance,

$$\varrho_{0,X}^{\mathsf{r}}(x \coloneqq e) = \lambda s.\, \mathsf{ret}_{s[x:=\mathsf{eev}(e,s)]} \qquad \text{and} \qquad \varrho_{0,X}^{\mathsf{w}}((c,u)\,;(q,f)) = \begin{cases} d\,;q & \text{if u = d;} \\ ((d\,;q), s) & \text{if u = (d, s);} \\ ([q]_s, s) & \text{if u = s.} \end{cases}$$

See [17, App. A] for the full definition of $\varrho_0$. We will next introduce trace, cost, and termination semantics for $B_0$-coalgebras (including the operational model of $\mathsf{Imp}^2$) and demonstrate that unlike the case of $D^{\mathcal{S}}$-coalgebras in the previous section, these notions can be directly captured by coalgebraic similarity for suitable choices of relation liftings and weakenings. In this way, we are able to derive the respective congruence properties for $\mathsf{Imp}^2$ using abstract GSOS theory.

[1]We use superscripts for the sorts of morphisms to reserve subscripts for components of natural transformations.

**Remark 3.2.** To model the various forms of semantics coalgebraically, we extend the behaviour functor $B_0$ to the 'nondeterministic' functor $B\colon \mathbf{Set}^2 \to \mathbf{Set}^2$ given by

$$B_{\mathsf{r}}X = X_{\mathsf{w}}^{\mathcal{S}} \qquad \text{and} \qquad B_{\mathsf{w}}X = \mathcal{P}(X_{\mathsf{w}} \times \mathcal{S} + X_{\mathsf{w}} + \mathcal{S}). \tag{15}$$

Note that $\mathcal{P}$ only appears in the writer sort. This allows us to work with weak transitions of writers. We tacitly identify a $B_0$-coalgebra $(X, \chi_0)$ with the deterministic $B$-coalgebra $(X, \chi)$ given by

$$\chi^{\mathsf{r}} = \chi_0^{\mathsf{r}} \qquad \text{and} \qquad \chi^{\mathsf{w}}(c) = \{\chi_0^{\mathsf{w}}(c)\}.$$

The GSOS law $\varrho_0$ of $\Sigma$ over $B_0$ extends to a GSOS law $\varrho$ of $\Sigma$ over $B$ by using $\varrho_0$ elementwise, e.g.,

$$\varrho_X^{\mathsf{w}}((c,U)\,;(q,f)) = \{d\,;q \mid d \in U\} \cup \{((d\,;q),s) \mid (d,s) \in U\} \cup \{([q]_s,s) \mid s \in U\}. \tag{16}$$

Since $\varrho$ carries the same information as $\varrho_0$, the operational models of $\varrho_0$ and $\varrho$ coincide modulo the above identification of $B_0$- and $B$-coalgebras.

### 3.2 Trace Semantics for Imp$^2$

To define trace semantics for $B_0$-coalgebras, we need to introduce suitable weak transitions:

**Notation 3.3.** Let $\chi\colon X \to BX$ be a $B$-coalgebra. For $p \in X_{\mathsf{r}}$, $c, d \in X_{\mathsf{w}}$ and $s \in \mathcal{S}$ we write

- $p, s \to c$ if $\chi^{\mathsf{r}}(p)(s) = c$, and $c \to d \,/\, c \xrightarrow{s} d \,/\, c \downarrow s$ if $d \,/\, (d,s) \,/\, s \in \chi^{\mathsf{w}}(c)$, respectively.
- $c \overset{1}{\Longrightarrow} d$ if there exist $n \geq 0$ and $c_0, \ldots, c_n \in X_{\mathsf{w}}$ with $c = c_0 \to c_1 \to \cdots \to c_n = d$.
- $c \overset{1,s}{\Longrightarrow} d$ if there exists $c' \in X_{\mathsf{w}}$ with $c \overset{1}{\Longrightarrow} c' \to d, s$;
- $c \Downarrow^1 s$ if there exists $c' \in X_{\mathsf{w}}$ with $c \overset{1}{\Longrightarrow} c' \downarrow s$.

The *trace map* of a $B_0$-coalgebra, i.e. a deterministic $B$-coalgebra $(X, \chi)$, is the two-sorted map $\mathrm{trc}\colon X \to ((\mathcal{S}^\infty)^{\mathcal{S}}, \mathcal{S}^\infty)$ defined as follows:

- $\mathrm{trc}^{\mathsf{r}}(p)(s) = \mathrm{trc}^{\mathsf{w}}(c)$ if $p, s \to c$;
- $\mathrm{trc}^{\mathsf{w}}(c) = (s_1, \ldots, s_n, s)$ if there exist $c = c_0, \ldots, c_n \in X_{\mathsf{w}}$ with $c_i \overset{1}{\Longrightarrow} c_{i+1}, s_{i+1}$ $(i < n)$ and $c_n \Downarrow^1 s$;
- $\mathrm{trc}^{\mathsf{w}}(c) = (s_1, s_2, s_3, \ldots)$ if there exist $c = c_0, c_1, c_2, \ldots \in X_{\mathsf{w}}$ with $c_i \overset{1}{\Longrightarrow} c_{i+1}, s_{i+1}$ for all $i \in \mathbb{N}$.

We say that $p, q \in X_{\mathsf{r}}$ are *trace equivalent* if $\mathrm{trc}^{\mathsf{r}}(p) = \mathrm{trc}^{\mathsf{r}}(q)$; similarly for $c, d \in X_{\mathsf{w}}$. To capture trace equivalence for $B_0$-coalgebras, we work with the extended functor $B$ (15) and choose a relation lifting of $B$ and a notion of weakening for $B$-coalgebras as follows:

*Relation lifting.* We take the relation lifting $\bar{B}_{\mathrm{trc}}\colon \mathbf{Rel}(\mathbf{Set}^2) \to \mathbf{Rel}(\mathbf{Set}^2)$ of $B$ defined by

$$(\bar{B}_{\mathrm{trc}}R)_{\mathsf{r}} = R_{\mathsf{w}}^{\mathcal{S}} \qquad \text{and} \qquad (\bar{B}_{\mathrm{trc}}R)_{\mathsf{w}} = \vec{\mathcal{P}}(R_{\mathsf{w}} \times \Delta_{\mathcal{S}} + R_{\mathsf{w}} + \Delta_{\mathcal{S}}). \tag{17}$$

Here $\vec{\mathcal{P}}$ is the asymmetric Egli-Milner lifting (Example 2.7), $\Delta_{\mathcal{S}}$ is the identity relation of $\mathcal{S}$, and $(-)^{\mathcal{S}}$, $\times$, $+$ are the $\mathcal{S}$-fold power, the product and the coproduct of single-sorted relations (Section 2.2).

*Weakening.* The weakening $\tilde{\chi}_{\mathrm{trc}}\colon X \to BX$ of a coalgebra $\chi\colon X \to BX$ is defined by

$$\tilde{\chi}_{\mathrm{trc}}^{\mathsf{r}} = \chi^{\mathsf{r}} \qquad \text{and} \qquad \tilde{\chi}_{\mathrm{trc}}^{\mathsf{w}}(c) = \{(d,s) \mid c \overset{1}{\Longrightarrow} d, s\} \cup \{d \mid c \overset{1}{\Longrightarrow} d\} \cup \{s \mid c \Downarrow^1 s\}. \tag{18}$$

See [17, App. B] for details as to why $\tilde{\chi}_{\mathrm{trc}}$ is a weakening.

For every $B$-coalgebra $(X, \chi)$, we thus obtain from Definition 2.12 the generic notions of $\bar{B}_{\mathrm{trc}}$-simulation and $\bar{B}_{\mathrm{trc}}$-similarity on $(X, \chi, \tilde{\chi}_{\mathrm{trc}})$. They spell out as follows:

**Definition 3.4.** A *trace simulation* on a $B$-coalgebra $(X, \chi)$ is a relation $R \rightarrowtail X \times X$ such that the following holds for all $p, q \in X_{\mathsf{r}}$, $c, d \in X_{\mathsf{w}}$ and $s \in \mathcal{S}$:

(1) If $R_{\mathsf{r}}(p, q)$ and $p, s \to c$ then there exists $d$ such that $q, s \to d$ and $R_{\mathsf{w}}(c, d)$.

(2) If $R_{\mathsf{w}}(c,d)$ and $c \to c'$ then there exists $d'$ such that $d \overset{1}{\Rightarrow} d'$ and $R_{\mathsf{w}}(c',d')$.

(3) If $R_{\mathsf{w}}(c,d)$ and $c \to c', s$ then there exists $d'$ such that $d \overset{1,s}{\Longrightarrow} d'$ and $R_{\mathsf{w}}(c',d')$.

(4) If $R_{\mathsf{w}}(c,d)$ and $c \downarrow s$ then $d \Downarrow^1 s$.

*Trace similarity* is the greatest trace simulation.

For $B_0$-coalgebras, this notion of simulation captures precisely trace equivalence:

**Proposition 3.5.** *For every $B_0$-coalgebra, trace similarity coincides with trace equivalence.*

### 3.3 Cost Semantics for Imp²

Given a $B_0$-coalgebra $\chi\colon X \to B_0X$, we define a two-sorted map $\mathrm{cst}\colon X \to ((\mathbb{N}\times\mathcal{S}+1)^{\mathcal{S}}, \mathbb{N}\times\mathcal{S}+1)$:

$$\mathrm{cst}^{\mathsf{r}}(p)(s) = \mathrm{cst}^{\mathsf{w}}(c) \quad \text{if} \quad p, s \to c, \qquad \mathrm{cst}^{\mathsf{w}}(c) = \begin{cases} (n,s) & \text{if } \mathrm{trc}^{\mathsf{w}}(c) = (s_1,\dots,s_n,s) \in \mathcal{S}^+, \\ * & \text{if } \mathrm{trc}^{\mathsf{w}}(c) \in \mathcal{S}^\omega. \end{cases}$$

We say that $p, q \in X_{\mathsf{r}}$ are *cost equivalent* if $\mathrm{cst}^{\mathsf{r}}(p) = \mathrm{cst}^{\mathsf{r}}(q)$; similarly for $c, d \in X_{\mathsf{w}}$. To capture cost equivalence for $B_0$-coalgebras, we work with a relation lifting and weakening chosen as follows:

*Relation lifting.* We take the relation lifting $\bar{B}_{\mathrm{cst}}$ of $B$ defined below, where $\top_{\mathcal{S}} = \mathcal{S}\times\mathcal{S}$:

$$(\bar{B}_{\mathrm{cst}}R)_{\mathsf{r}} = R_{\mathsf{w}}^{\mathcal{S}} \qquad \text{and} \qquad (\bar{B}_{\mathrm{cst}}R)_{\mathsf{w}} = \vec{\mathcal{P}}(R_{\mathsf{w}} \times \top_{\mathcal{S}} + R_{\mathsf{w}} + \Delta_{\mathcal{S}}). \tag{19}$$

Thus, in comparison to $\bar{B}_{\mathrm{trc}}$, the first occurrence of $\Delta_{\mathcal{S}}$ in the writer sort is replaced with $\top_{\mathcal{S}}$.

*Weakening.* We take the same weakening as for trace semantics: $\tilde{\chi}_{\mathrm{cst}} = \tilde{\chi}_{\mathrm{trc}}$. See [17, App. B] for details as to why $\tilde{\chi}_{\mathrm{cst}}$ is a weakening with respect to $\bar{B}_{\mathrm{cst}}$.

Given a $B$-coalgebra $(X,\chi)$, the notion of $\bar{B}_{\mathrm{cst}}$-simulation on $(X,\chi,\tilde{\chi}_{\mathrm{cst}})$ now gives:

**Definition 3.6.** A *cost simulation* on a $B$-coalgebra $(X,\chi)$ is a relation $R \rightarrowtail X\times X$ such that the following holds for all $p, q \in X_{\mathsf{r}}$, $c, d \in X_{\mathsf{w}}$ and $s \in \mathcal{S}$:

(1) If $R_{\mathsf{r}}(p,q)$ and $p, s \to c$ then there exists $d$ such that $q, s \to d$ and $R_{\mathsf{w}}(c,d)$.

(2) If $R_{\mathsf{w}}(c,d)$ and $c \to c'$ then there exists $d'$ such that $d \overset{1}{\Rightarrow} d'$ and $R_{\mathsf{w}}(c',d')$.

(3) If $R_{\mathsf{w}}(c,d)$ and $c \xrightarrow{s} c'$ then there exist $d', s'$ such that $d \overset{1,s'}{\Longrightarrow} d'$ and $R_{\mathsf{w}}(c',d')$.

(4) If $R_{\mathsf{w}}(c,d)$ and $c \downarrow s$ then $d \Downarrow^1 s$.

*Cost similarity* is the greatest cost simulation.

The only difference to trace simulations lies in condition (3): due the more permissive lifting, it suffices to match $c \xrightarrow{s} c'$ with a weak transition $d \overset{1,s'}{\Longrightarrow} d'$ where not necessarily $s = s'$. However, cost similar terms still produce the same *number* of states, so we get:

**Proposition 3.7.** *For every $B_0$-coalgebra, cost similarity coincides with cost equivalence.*

### 3.4 Termination Semantics for Imp²

Given a $B_0$-coalgebra $\chi\colon X \to B_0X$, we define the two-sorted map $\mathrm{ter}\colon X \to ((\mathcal{S}+1)^{\mathcal{S}}, \mathcal{S}+1)$ by

$$\mathrm{ter}^{\mathsf{r}}(p)(s) = \mathrm{ter}^{\mathsf{w}}(c) \quad \text{if} \quad p, s \to c, \qquad \mathrm{ter}^{\mathsf{w}}(c) = \begin{cases} s & \text{if } \mathrm{trc}^{\mathsf{w}}(c) = (s_1,\dots,s_n,s), \\ * & \text{if } \mathrm{trc}^{\mathsf{w}}(c) \in \mathcal{S}^\omega. \end{cases}$$

We say that $p, q \in X_{\mathsf{r}}$ are *termination equivalent* if $\mathrm{ter}^{\mathsf{r}}(p) = \mathrm{ter}^{\mathsf{r}}(q)$; similarly for $c, d \in X_{\mathsf{w}}$. We capture termination equivalence via relation liftings and weakenings as follows:

*Relation lifting.* We take the same relation lifting as for cost semantics:

$$\bar{B}_{\mathsf{ter}} = \bar{B}_{\mathsf{cst}}. \tag{20}$$

*Weakening.* We extend the weak transitions $\overset{1}{\Rightarrow}, \Downarrow^1$ of Notation 3.3 to a more liberal version:

**Notation 3.8.** Given a coalgebra $\chi\colon X \to BX$ and $c, d \in X_{\mathsf{w}}$, $s \in \mathcal{S}$, we write

- $c \overset{2}{\Rightarrow} d$ and $c \overset{2,s}{\Longrightarrow} d$ if there exist $n \geq 0$ and $c = c_0, \ldots, c_n = d \in X_{\mathsf{w}}$ such that, for each $i < n$,

$$c_i \to c_{i+1} \qquad \text{or} \qquad \exists s_{i+1}.\, c_i \xrightarrow{s_{i+1}} c_{i+1}.$$

- $c \Downarrow^2 s$ if there exists $c' \in X_{\mathsf{w}}$ such that $c \overset{2}{\Rightarrow} c' \downarrow s$.

The weak transitions $\overset{2}{\Rightarrow}, \Downarrow^2$ thus completely ignore the intermediate states produced in a sequence of strong transitions. We define the weakening $\tilde{\chi}_{\mathsf{ter}}\colon X \to BX$ of $\chi$ by

$$\tilde{\chi}^{\mathsf{r}}_{\mathsf{ter}} = \chi^{\mathsf{r}} \qquad \text{and} \qquad \tilde{\chi}^{\mathsf{w}}_{\mathsf{ter}}(c) = \{(d, s) \mid c \overset{2,s}{\Longrightarrow} d\} \cup \{d \mid c \overset{2}{\Rightarrow} d\} \cup \{s \mid c \Downarrow^2 s\}. \tag{21}$$

See [17, App. B] for details as to why $\tilde{\chi}_{\mathsf{ter}}$ is a weakening with respect to $\bar{B}_{\mathsf{ter}}$.

For every $B$-coalgebra $(X, \chi)$, the notion of $\bar{B}_{\mathsf{ter}}$-simulation on $(X, \chi, \tilde{\chi}_{\mathsf{ter}})$ then corresponds to:

**Definition 3.9.** A *termination simulation* on a $B$-coalgebra $(X, \chi)$ is a relation $R \rightarrowtail X \times X$ such that the following holds for all $p, q \in X_{\mathsf{r}}$, $c, d \in X_{\mathsf{w}}$ and $s \in \mathcal{S}$:

(1) If $R_{\mathsf{r}}(p, q)$ and $p, s \to c$ then there exists $d$ such that $q, s \to d$ and $R_{\mathsf{w}}(c, d)$.

(2) If $R_{\mathsf{w}}(c, d)$ and $c \xrightarrow{s} c'$ or $c \to c'$, then there exists $d'$ such that $d \overset{2}{\Rightarrow} d'$ and $R_{\mathsf{w}}(c', d')$.

(3) If $R_{\mathsf{w}}(c, d)$ and $c \downarrow s$ then $d \Downarrow^2 s$.

*Termination similarity* is the greatest termination simulation and is denoted by $\preceq_{\mathsf{ter}}$.

Since termination similarity only observes terminating states, we get:

**Proposition 3.10.** *For every $B_0$-coalgebra, the relation $\preceq_{\mathsf{ter}} \cap \succeq_{\mathsf{ter}}$ equals termination equivalence.*

### 3.5 Compositionality of $\mathsf{Imp}^2$ and Imp

We have shown that trace, cost, and termination equivalence for $B_0$-coalgebras correspond to coalgebraic similarity for suitable choices of relations liftings and weakenings. We now instantiate the general compositionality result for abstract GSOS (Theorem 2.20) to the AOS

$$O_\star = (\Sigma, \overline{\Sigma}, B, \bar{B}_\star, \varrho, \gamma, \tilde{\gamma}_\star) \qquad (\star \in \{\mathsf{trc}, \mathsf{cst}, \mathsf{ter}\}),$$

where the functor $B$ is ordered by $f \preceq g$ iff $f^{\mathsf{r}} = g^{\mathsf{r}}$ and $\forall z \in Z_{\mathsf{w}}.\, f^{\mathsf{w}}(z) \subseteq g^{\mathsf{w}}(z)$ for $f, g\colon Z \to BX$, and moreover $\varrho$ is the extended GSOS law (16) for $\mathsf{Imp}^2$ and $\gamma$ is its operational model. This yields:

**Theorem 3.11** (Compositionality of $\mathsf{Imp}^2$)**.** *Trace equivalence, cost equivalence and termination equivalence form a congruence on the operational model of $\mathsf{Imp}^2$.*

Proof sketch. We only need to verify the conditions (1)–(3) of Theorem 2.20. We illustrate the case of trace semantics, the other two being almost identical. Put $\bar{B} = \bar{B}_{\mathsf{trc}}$ and $\tilde{\gamma} = \tilde{\gamma}_{\mathsf{trc}}$.

(1) Up-closure and preservation of composition and identities by $\bar{B}$ follow by an easy computation.

(2) We need to show that the GSOS law $\varrho$ is liftable, that is, for each $R \rightarrowtail X \times X$ in $\mathbf{Set}^2$ the map $\varrho_X\colon \Sigma(X \times BX) \to B\Sigma^\star X$ is a relation morphism from $\overline{\Sigma}(R \times \bar{B}R)$ to $\bar{B}\overline{\Sigma}^\star R$. This boils down to observing that the rules of $\mathsf{Imp}^2$ (Figure 2) are parametrically polymorphic. Let us give the argument for the operator ';' of arity $\mathsf{w} \times \mathsf{r} \to \mathsf{r}$. Consider two elements $(c, U)\,;(q, f)$ and $(c', U')\,;(q', f')$ of

$\Sigma_{\mathsf{w}}(X \times BX)$ that are related in $(\overline{\Sigma}(R \times \bar{B}R))_{\mathsf{w}}$; in particular, $(U, U') \in \vec{\mathcal{P}}(R_{\mathsf{w}} \times \Delta_{\mathcal{S}} + R_{\mathsf{w}} + \Delta_{\mathcal{S}})$ and $(q, q') \in R_{\mathsf{r}}$. Since

$$\varrho_X^{\mathsf{w}}((c,U)\,;(q,f)) = \{d\,;\, q \mid d \in U\} \cup \{((d\,;\,q),s) \mid (d,s) \in U\} \cup \{([q]_s, s) \mid s \in U\},$$
$$\varrho_X^{\mathsf{w}}((c',U')\,;(q',f')) = \{d'\,;\, q' \mid d' \in U'\} \cup \{((d'\,;\,q'),s) \mid (d',s) \in U'\} \cup \{([q']_s, s) \mid s \in U'\}$$

by (16), it immediately follows that

$$(\varrho_X^{\mathsf{w}}((c,U)\,;(q,f)),\ \varrho_X^{\mathsf{w}}((c',U')\,;(q',f'))) \in \vec{\mathcal{P}}((\overline{\Sigma}^* R)_{\mathsf{w}} \times \Delta_{\mathcal{S}} + (\overline{\Sigma}^* R)_{\mathsf{w}} + \Delta_{\mathcal{S}}) = (\bar{B}\overline{\Sigma}^\star R)_{\mathsf{w}}, \quad (22)$$

as required. The other operators are treated analogously.

(3) Finally, we show that $(\mu\Sigma, \gamma, \tilde{\gamma})$ is a lax $\varrho$-bialgebra. By (8) and the definition of $\tilde{\gamma}$, this means that the writer rules of Figure 2 remain sound in the operational model $\gamma\colon (\mathsf{R},\mathsf{W}) \to B(\mathsf{R},\mathsf{W})$ when strong transitions $\to, \downarrow$ are replaced with corresponding weak transitions $\overset{1}{\Rightarrow}, \Downarrow^1$. For illustration, let us consider two of the writer rules for sequential composition and their weak versions:

$$\frac{c \to d}{c\,;q \to d\,;q} \qquad \frac{c \overset{s}{\to} d}{c\,;q \overset{s}{\to} d\,;q} \qquad \frac{c \overset{1}{\Rightarrow} d}{c\,;q \overset{1}{\Rightarrow} d\,;q} \qquad \frac{c \overset{1,s}{\Longrightarrow} d}{c\,;q \overset{1,s}{\Longrightarrow} d\,;q}$$

Soundness of the third and fourth rule means that if the premise holds, then the conclusion holds. The third rule is sound because it follows by repeated application of the first one. The fourth rule is sound because it emerges from the third rule followed by one application of the second rule. □

It remains to relate the compositionality of $\mathsf{Imp}^2$ to that of the original language Imp. To this end, we make the key observation that the embedding of Imp into $\mathsf{Imp}^2$ is *semantics-preserving*: the trace of an Imp-term is the same regardless of whether it is executed as a program of Imp or as a reader of $\mathsf{Imp}^2$. In the following we denote the trace maps of Imp and $\mathsf{Imp}^2$ by $\mathsf{trc}_{\mathsf{Imp}}\colon \mathsf{P} \to (\mathcal{S}^\infty)^{\mathcal{S}}$ and $\mathsf{trc}_{\mathsf{Imp}^2}\colon (\mathsf{R},\mathsf{W}) \to ((\mathcal{S}^\infty)^{\mathcal{S}}, \mathcal{S}^\infty)$; recall that $\mathsf{P} = \mathsf{R}$. Similarly for the cost and termination maps.

**Theorem 3.12.** *The trace, cost, and termination semantics for* Imp *and* $\mathsf{Imp}^2$ *coincide:*

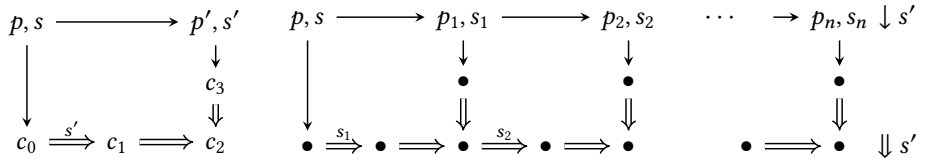
$$\mathsf{trc}_{\mathsf{Imp}} = \mathsf{trc}^r_{\mathsf{Imp}^2}, \qquad \mathsf{cst}_{\mathsf{Imp}} = \mathsf{cst}^r_{\mathsf{Imp}^2}, \qquad \mathsf{ter}_{\mathsf{Imp}} = \mathsf{ter}^r_{\mathsf{Imp}^2}.$$

Proof sketch. It suffices to prove the first equality; the other two are then immediate because cost and termination semantics in both Imp and $\mathsf{Imp}^2$ are derived from trace semantics. The key to the proof lies in relating transitions in the operational model (1) of Imp to (weak) transitions $\to$, $\overset{1}{\Rightarrow}, \Downarrow^1$ (Notation 3.3) in the operational model (13) of $\mathsf{Imp}^2$. We write $\Rightarrow, \Downarrow$ for $\overset{1}{\Rightarrow}, \Downarrow^1$. By structural induction one can show that the following holds for all $p, p' \in \mu\Theta$ and $s, s' \in \mathcal{S}$:

(1) If $p, s \to p', s'$ in Imp, then there exist $c_0, c_1, c_2, c_3 \in \mathsf{W}$ and transitions in $\mathsf{Imp}^2$ as depicted below on the left. (The upper arrow is an Imp-transition, the remaining arrows are $\mathsf{Imp}^2$-transitions.)

(2) If $p, s \downarrow s'$ in Imp, then there exists $c \in \mathsf{W}$ such that $p, s \to c$ and $c \Downarrow s'$ in $\mathsf{Imp}^2$.

The statement of the theorem now easily follows. Suppose that $\mathsf{trc}_{\mathsf{Imp}}(p)(s) = (s_1, \dots, s_n, s')$, witnessed by Imp-transitions $p, s \to p_1, s_1 \to p_2, s_2 \to \cdots \to p_n, s_n \downarrow s'$. Applying (1) to the first $n$ transitions and (2) to the last one yields the situation below on the right, where the $\bullet$'s are elements of $\mathsf{W}$, the transitions in the upper row are in Imp, and the remaining transitions are in $\mathsf{Imp}^2$.



Thus $\mathrm{trc}^{\mathsf{r}}_{\mathsf{Imp}^2}(p)(s) = (s_1, \ldots, s_n, s') = \mathrm{trc}_{\mathsf{Imp}}(p)(s)$. The case where $\mathrm{trc}_{\mathsf{Imp}}(p)(s)$ is an infinite trace is treated analogously. We conclude that $\mathrm{trc}_{\mathsf{Imp}} = \mathrm{trc}^{\mathsf{r}}_{\mathsf{Imp}^2}$ as claimed. □

In particular, two Imp-programs are trace/cost/termination equivalent iff they are trace/cost/termination equivalent as $\mathsf{Imp}^2$-readers, and so we recover the essential parts of Theorem 2.3:

**Theorem 3.13** (Compositionality of Imp)**.** *Trace equivalence, cost equivalence, and termination equivalence form congruences on the operational model of* Imp.

It turns out that our above results on $\mathsf{Imp}^2$ and its tight relation to Imp are not an accident: they extend to a whole family of stateful languages emerging from the general setting of *stateful SOS*, and in fact are arguably best understood at that level of abstraction.

## 4 From Stateful SOS to Abstract GSOS

In this section we revisit the developments for Imp and $\mathsf{Imp}^2$ at the level of *stateful SOS*, a general rule format for modelling stateful languages introduced by Goncharov et al. [15]. In particular, we will recover their main compositionality result by employing the methodology of abstract GSOS.

### 4.1 Stateful SOS Specifications

We start by recalling the definition of the stateful SOS format. Let us first settle some notation:

**Notation 4.1.** (1) We fix a single-sorted algebraic signature $\Theta$; as before, we also denote the induced polynomial functor by $\Theta\colon \mathbf{Set} \to \mathbf{Set}$. Moreover, we fix a set $\mathcal{S}$ of *states*. We think of $\Theta$ as the signature of a single-sorted stateful language such as Imp and of $\mathcal{S}$ as its set of variable stores.
(2) We continue to work with the functors

$$D\colon \mathbf{Set} \to \mathbf{Set}, \qquad DX = X \times \mathcal{S} + \mathcal{S},$$
$$B\colon \mathbf{Set}^2 \to \mathbf{Set}^2, \qquad B_{\mathsf{r}}X = X_{\mathsf{w}}^{\mathcal{S}}, \qquad B_{\mathsf{w}}X = \mathcal{P}(X_{\mathsf{w}} \times \mathcal{S} + X_{\mathsf{w}} + \mathcal{S}).$$

(3) Moreover, we fix a countably infinite set of variables

$$\mathcal{V} = \{x_n \mid n \in \mathbb{N}\} \cup \{y_n \mid n \in \mathbb{N}\}.$$

**Definition 4.2.** (1) A *stateful SOS* rule for an $n$-ary operator $\mathsf{f} \in \Theta$ is an expression of either form

$$\frac{x_i, s \to y_i, s_i \ (i \in W) \quad x_j, s \downarrow s_j \ (j \in \overline{W})}{\mathsf{f}(x_1, \ldots, x_n), s \to t, s'} \tag{23}$$

$$\frac{x_i, s \to y_i, s_i \ (i \in W) \quad x_j, s \downarrow s_j \ (j \in \overline{W})}{\mathsf{f}(x_1, \ldots, x_n), s \downarrow s'} \tag{24}$$

where $W \subseteq \{1, \ldots, n\}$, $\overline{W} = \{1, \ldots, n\} \setminus W$, $s, s', s_i \in \mathcal{S}$ for $i = 1, \ldots, n$ and $t \in \Sigma^\star\mathcal{V}$ is a term in the variables $x_1, \ldots, x_n$ and $y_i$ $(i \in W)$. The tuple $(W, s, s_1, \ldots, s_n)$ is the *trigger* of the rule.
(2) A *stateful SOS specification* is a set $\mathcal{L}$ of stateful SOS rules such that for each $n$-ary $\mathsf{f} \in \Theta$ and each $W \subseteq \{1, \ldots, n\}$, $s, s_1, \ldots, s_n \in \mathcal{S}$, there is a unique rule for f with trigger $(W, s, s_1, \ldots, s_n)$ in $\mathcal{L}$.

**Remark 4.3.** Stateful SOS specifications are in a bijective correspondence with *stateful SOS laws*, which are natural transformations of the form

$$\delta_X\colon \Theta(X \times DX) \times \mathcal{S} \to D(\Theta^\star X) \qquad (X \in \mathbf{Set}).$$

Similar to a GSOS law, a stateful SOS law simply encodes the rules of its corresponding stateful SOS specification into a natural family of functions; see Goncharov et al. [15] for more details.

**Notation 4.4.** Unused premises can be dropped as expected: given an $n$-ary operator f and disjoint subsets $W_0, W_1 \subseteq \{1, \dots .n\}$, the rule on the left below represents the set of all rules on the right where $W \subseteq \{1, \dots, n\}$ satisfies $W_0 \subseteq W$ and $W_1 \subseteq \overline{W}$ and $s_i \in \mathcal{S}$ for $i \in \{1, \dots, n\} \setminus (W_0 \cup W_1)$.

$$\frac{x_i, s \to y_i, s_i \ (i \in W_0) \quad x_j, s \downarrow s_j \ (j \in W_1)}{\mathsf{f}(x_1, \dots, x_n), s \to t, s'} \qquad \frac{x_i, s \to y_i, s_i \ (i \in W) \quad x_j, s \downarrow s_j \ (j \in \overline{W})}{\mathsf{f}(x_1, \dots, x_n), s \to t, s'}$$

Note that $t$ is then a term in the variables $x_1, \dots, x_n$ and $y_i$ for $i \in W_0$ and that $t$ and $s'$ only depend on $s$ and $s_i$ for $i \in W_0 \cup W_1$. Analogously for rules of the form (24). The special case $W_0 = W_1 = \emptyset$ corresponds to premise-free rules, where the behaviour of f is fully determined by the current state:

$$\frac{}{\mathsf{f}(x_1, \dots, x_n), s \to t, s'} \qquad \frac{}{\mathsf{f}(x_1, \dots, x_n), s \downarrow s'} \tag{25}$$

We say that an operator f is *passive* if all rules for f are premise-free. An *active* operator is one that is not passive, i.e. its behaviour actually depends on the behaviour of some of its operands.

A stateful SOS specification completely and deterministically defines the behaviour of each closed $\Theta$-term $\mathsf{f}(p_1, \dots, p_n)$ depending on the current state and the possible behaviours of its subterms $p_i$ on that state. Its operational model is the coalgebra that runs $\Theta$-terms according to the rules:

**Definition 4.5.** (1) Every stateful SOS specification $\mathcal{L}$ induces a GSOS law $\varrho^{\mathcal{L}}$ of $\Theta$ over $D^{\mathcal{S}}$ with

$$\varrho^{\mathcal{L}}_X \colon \Theta(X \times (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}) \to (\Sigma^\star X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$$

defined as follows: given an $n$-ary $\mathsf{f} \in \Theta$, $p_1, \dots, p_n \in X$, $u_1, \dots, u_n \in (X \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}$, and $s \in \mathcal{S}$, put

$$W = \{i \in \{1, \dots, n\} \mid u_i(s) \in X \times \mathcal{S}\}, \qquad (t_i, s_i) = u_i(s), \ \ i \in W, \qquad s_j = u_j(s), \ \ j \in \overline{W}.$$

Consider the unique rule in $\mathcal{L}$ for f with trigger $(W, s, s_1, \dots, s_n)$. If the rule is given by (23), put

$$\varrho^{\mathcal{L}}_X(\mathsf{f}((p_1, u_1), \dots, (p_n, u_n)))(s) = (t[p_1/x_1, \dots, p_n/x_n, t_{i_1}/y_{i_1}, \dots, t_{i_k}/y_{i_k}], s')$$

where $W = \{i_1, \dots, i_k\}$ and $-/-$ denotes substitution. If the rule is given by (24), put

$$\varrho^{\mathcal{L}}_X(\mathsf{f}((p_1, u_1), \dots, (p_n, u_n)))(s) = s'.$$

(2) The *operational model* of $\mathcal{L}$ is the operational model of the GSOS law $\varrho^{\mathcal{L}}$, denoted by

$$\gamma^{\mathcal{L}} \colon \mu\Theta \to (\mu\Theta \times \mathcal{S} + \mathcal{S})^{\mathcal{S}}. \tag{26}$$

**Example 4.6.** The rules of Imp (Figure 1) form a stateful SOS specification $\mathcal{L}$ for the signature $\Theta$ of Imp, modulo renaming of variables and dropping unused premises (Notation 4.4). For instance, the rule for sequential composition shown on the left represents the rules on the right for all $s_2 \in \mathcal{S}$.

$$\frac{p, s \downarrow s'}{p\,;q, s \to q, s'} \qquad \frac{x_1, s \downarrow s' \quad x_2, s \to y_2, s_2}{x_1\,;x_2, s \to x_2, s'} \qquad \frac{x_1, s \downarrow s' \quad x_2, s \downarrow s_2}{x_1\,;x_2, s \to x_2, s'}$$

The induced GSOS law $\varrho^{\mathcal{L}}$ is that of Example 2.16, and the operational model of $\mathcal{L}$ is the coalgebra (1) that runs Imp-terms according the rules of the language.

### 4.2 Reader-Writer Semantics for Stateful SOS

We will now generalize the transformation of Imp into Imp$^2$ to the level of stateful SOS specifications. This requires a syntactic restriction of the rules:

**Definition 4.7.** A stateful SOS specification is *cool* if for every $n$-ary active operator f there exists $j \in \{1, \dots, n\}$ (the *receiving position of* f) such that all rules for f are of one of the following forms:

$$\frac{x_j, s \to y_j, s'}{\mathsf{f}(x_1, \dots, x_{j-1}, x_j, x_{j+1}, \dots, x_n), s \to \mathsf{f}(x_1, \dots, x_{j-1}, y_j, x_{j+1}, \dots, x_n), s'} \tag{27}$$

$$\frac{x_j, s \downarrow s'}{\mathsf{f}(x_1, \dots, x_n), s \to t, s''} \qquad \frac{x_j, s \downarrow s'}{\mathsf{f}(x_1, \dots, x_n), s \downarrow s''} \tag{28}$$

where in (28) we have $t \in \Sigma^\star(\{x_1, \dots, x_n\} \setminus \{x_j\})$, and $t$ and $s''$ depend only on $s'$ but not on $s$.

The cool format asserts that an active operator f runs its $j$-th subterm until termination and then discards it, proceeding to a state derivable from the terminating state of the subterm. The terminology alludes to the *cool* congruence formats for labelled transition systems by Bloom [11] and van Glabbeek [46]. In particular, rules of type (27) are a stateful version of *patience rules* [46].

**Example 4.8.** The language Imp (Figure 1) is cool. Its only active operator is sequential composition $-\,;-$, which is receiving in the first position.

**Notation 4.9.** For the remainder of this section, we fix a cool stateful SOS specification $\mathcal{L}$ over the signature $\Theta$. We use the following notation for its passive and active operators:

- We let $\Theta_{\mathrm{pas},n}$ denote the set of $n$-ary passive operators. For $\mathsf{f} \in \Theta_{\mathrm{pas},n}$ and $s \in \mathcal{S}$ we put $o(\mathsf{f}, s) = t, s'$ or $o(\mathsf{f}, s) = s'$ if $\mathcal{L}$ contains the first or second rule (25), respectively.
- We let $\Theta_{\mathrm{act},n,j}$ denote the set of $n$-ary active operators with receiving position $j$. For $\mathsf{f} \in \Theta_{\mathrm{act},n,j}$ and $s, s' \in \mathcal{S}$, we put $o(\mathsf{f}, s, s') = t, s''$ or $o(\mathsf{f}, s, s') = s''$ if $\mathcal{L}$ contains the first or second rule (28).

The signature $\Theta$ extends to the two-sorted signature $\Sigma$ (with sorts r and w) containing the operators

- $\mathsf{f} \colon \mathsf{r} \times \cdots \times \mathsf{r} \to \mathsf{r}$ (with $n$ inputs) for every $n$-ary $\mathsf{f} \in \Theta$;
- $\overline{\mathsf{f}} \colon \mathsf{r} \times \cdots \times \mathsf{r} \times \mathsf{w} \times \mathsf{r} \times \cdots \times \mathsf{r} \to \mathsf{w}$ (with $n$ inputs and w in the $j$-th position) for every $\mathsf{f} \in \Theta_{\mathrm{act},n,j}$.
- a constant $\mathrm{ret}_s \colon w$ and unary operators $[-]_s \colon \mathsf{r} \to \mathsf{w}$ and $s.- \colon \mathsf{w} \to \mathsf{w}$ for every $s \in \mathcal{S}$.

The signature $\Sigma$ corresponds to the polynomial functor $\Sigma \colon \mathbf{Set}^2 \to \mathbf{Set}^2$ given by

$$\Sigma_{\mathsf{r}} X = \Theta X_{\mathsf{r}} \quad \text{and} \quad \Sigma_{\mathsf{w}} X = \underbrace{X_{\mathsf{r}} \times \mathcal{S}}_{[-]_s} + \underbrace{\mathcal{S}}_{\mathrm{ret}_s} + \underbrace{\mathcal{S} \times X_{\mathsf{w}}}_{s.-} + \coprod_{n,j} \coprod_{\mathsf{f} \in \Theta_{\mathrm{act},n,j}} \underbrace{X_{\mathsf{r}}^{j-1} \times X_{\mathsf{w}} \times X_{\mathsf{r}}^{n-j}}_{\overline{\mathsf{f}}}.$$

Elements of $(\mu\Sigma)_{\mathsf{r}}$ and $(\mu\Sigma)_{\mathsf{w}}$ are called *readers* and *writers*, respectively. Note that

$$(\mu\Sigma)_{\mathsf{r}} = \mu\Theta.$$

For the case where $\Theta$ is the single-sorted signature of Imp, the induced $\Sigma$ is the two-sorted signature of $\mathsf{Imp}^2$. The refinement of the operational rules of Imp to the reader-writer rules of $\mathsf{Imp}^2$ (Figure 2) then generalizes to the present setting as follows:

**Definition 4.10.** We fix a new set of variables

$$\mathcal{V}^2 = \{x_n \mid n \in \mathbb{N}\} \uplus \{v_n \mid n \in \mathbb{N}\} \uplus \{w_n \mid n \in \mathbb{N}\}$$

with $x_n$ representing readers and $v_n, w_n$ representing writers. (We use two types of writer variables for convenience, so that writer transitions in premises of rules can be written as $v_j \to w_j$ or $v_j \xrightarrow{s} w_j$.) The *reader-writer extension* $\mathcal{L}^2$ of $\mathcal{L}$ is given by the operational rules of Figure 3. Its *operational model* is the $B$-coalgebra

$$\gamma^{\mathcal{L}^2} \colon \mu\Sigma \to ((\mu\Sigma)_{\mathsf{w}}^{\mathcal{S}}, \mathcal{P}((\mu\Sigma)_{\mathsf{w}} \times \mathcal{S} + (\mu\Sigma)_{\mathsf{w}} + \mathcal{S})) \tag{29}$$

that runs terms according to the above rules.

Analogous to the special case $\mathsf{Imp}^2$, the operational model of $\mathcal{L}^2$ is actually deterministic (that is, it can be identified with a $B_0$-coalgebra), and it forms the operational model of a GSOS law $\varrho^{\mathcal{L}^2}$ of $\Sigma$ over $B$ that encodes the rules of $\mathcal{L}^2$. For the full definition of $\varrho^{\mathcal{L}^2}$, see [17, App. A]. As an alternative to the present rule-based definition, it is also possible to construct the law $\varrho^{\mathcal{L}^2}$ diagrammatically from $\varrho^{\mathcal{L}}$ using functorial strength.

$$\frac{\mathsf{f} \in \Theta_{\mathsf{pas},n} \quad o(\mathsf{f},s) = t, s'}{\mathsf{f}(x_1,\dots,x_n), s \to s'.[t]_{s'}}\ (\mathrm{f1}) \qquad \frac{\mathsf{f} \in \Theta_{\mathsf{pas},n} \quad o(\mathsf{f},s) = s'}{\mathsf{f}(x_1,\dots,x_n), s \to \mathsf{ret}_{s'}}\ (\mathrm{f2})$$

$$\frac{\mathsf{f} \in \Theta_{\mathsf{act},n,j}}{\mathsf{f}(x_1,\dots,x_j,\dots,x_n), s \to \bar{\mathsf{f}}(x_1,\dots,[x_j]_s,\dots,x_n)}\ (\mathrm{f3})$$

$$\frac{\mathsf{f} \in \Theta_{\mathsf{act},n,j} \quad v_j \to w_j}{\bar{\mathsf{f}}(x_1,\dots,v_j,\dots,x_n) \to \bar{\mathsf{f}}(x_1,\dots,w_j,\dots,x_n)}\ (\bar{\mathrm{f}}1) \qquad \frac{\mathsf{f} \in \Theta_{\mathsf{act},n,j} \quad o(\mathsf{f},s,s') = t, s'' \quad v_j \downarrow s'}{\bar{\mathsf{f}}(x_1,\dots,v_j,\dots,x_n) \xrightarrow{s''} [t]_{s''}}\ (\bar{\mathrm{f}}3)$$

$$\frac{\mathsf{f} \in \Theta_{\mathsf{act},n,j} \quad v_j \xrightarrow{s} w_j}{\bar{\mathsf{f}}(x_1,\dots,v_j,\dots,x_n) \xrightarrow{s} \bar{\mathsf{f}}(x_1,\dots,w_j,\dots,x_n)}\ (\bar{\mathrm{f}}2) \qquad \frac{\mathsf{f} \in \Theta_{\mathsf{act},n,j} \quad o(\mathsf{f},s,s') = s'' \quad v_j \downarrow s'}{\bar{\mathsf{f}}(x_1,\dots,v_j,\dots,x_n) \downarrow s''}\ (\bar{\mathrm{f}}4)$$

$$\frac{x_1, s \to v_1}{[x_1]_s \to v_1}\ ([-]) \qquad \frac{}{\mathsf{ret}_s \downarrow s}\ (\mathrm{ret}) \qquad \frac{}{s.v_1 \xrightarrow{s} v_1}\ (s.-)$$

Fig. 3. Reader semantics of $\mathcal{L}^2$ (top) and writer semantics of $\mathcal{L}^2$ (bottom).

### 4.3 Compositionality of $\mathcal{L}^2$ and $\mathcal{L}$

We now derive compositionality of the reader-writer extension $\mathcal{L}^2$ of $\mathcal{L}$ with respect to trace, cost, and termination semantics as an application of the general compositionality result for abstract GSOS (Theorem 2.20). To this end, we instantiate the theorem to the AOS

$$\mathcal{O}_\star = (\Sigma, \overline{\Sigma}, B, \bar{B}_\star, \varrho^{\mathcal{L}^2}, \gamma^{\mathcal{L}^2}, \tilde{\gamma}_\star^{\mathcal{L}^2}) \qquad (\star \in \{\mathrm{trc}, \mathrm{cst}, \mathrm{ter}\}),$$

with the relation liftings $\bar{B}_\star$ of $B$ given by (17), (19), (20) and the weakenings $\tilde{\gamma}_\star^{\mathcal{L}^2}$ of the operational model $\gamma^{\mathcal{L}^2}$ given by (18), (19), (20). Recall that $\bar{B}_{\mathrm{trc}}$-similarity on $(\mu\Sigma, \gamma^{\mathcal{L}^2}, \tilde{\gamma}^{\mathcal{L}^2})$ is trace equivalence (Proposition 3.5), $\bar{B}_{\mathrm{cst}}$-similarity is cost equivalence (Proposition 3.7), and the relation $\preceq_{\mathrm{ter}} \cap \succeq_{\mathrm{ter}}$ (where $\preceq_{\mathrm{ter}}$ denotes $\bar{B}_{\mathrm{ter}}$-similarity) is termination equivalence (Proposition 3.10). Consequently:

**Theorem 4.11** (Compositionality of $\mathcal{L}^2$)**.** *Trace equivalence, cost equivalence and termination equivalence form congruences on the operational model of $\mathcal{L}^2$.*

To prove the theorem, we only need to show that the conditions of Theorem 2.20 hold. The arguments are similar to the proof sketch of Theorem 3.11, with the coolness restriction on the given stateful SOS specification $\mathcal{L}$ being the key to the lax bialgebra condition.

Lastly, we show that congruence properties of the original language $\mathcal{L}$ can be reduced to those of $\mathcal{L}^2$. We denote the trace, cost, and termination maps on the operational models of $\mathcal{L}$ and $\mathcal{L}^2$ by

$$\begin{aligned} &\mathrm{trc}_{\mathcal{L}} \colon \mu\Theta \to (\mathcal{S}^\infty)^{\mathcal{S}}, && \mathrm{trc}_{\mathcal{L}^2} \colon \mu\Sigma \to ((\mathcal{S}^\infty)^{\mathcal{S}}, \\ &\mathrm{cst}_{\mathcal{L}} \colon \mu\Theta \to (\mathbb{N} \times \mathcal{S} + 1)^{\mathcal{S}}, && \mathrm{cst}_{\mathcal{L}^2} \colon \mu\Sigma \to ((\mathbb{N} \times \mathcal{S} + 1)^{\mathcal{S}}, \\ &\mathrm{ter}_{\mathcal{L}} \colon \mu\Theta \to (\mathcal{S} + 1)^{\mathcal{S}}, && \mathrm{ter}_{\mathcal{L}^2} \colon \mu\Sigma \to ((\mathcal{S} + 1)^{\mathcal{S}}, \mathcal{S} + 1). \end{aligned}$$

Generalizing Theorem 3.12, the extension of $\mathcal{L}$ to $\mathcal{L}^2$ leaves the semantics unchanged:

**Theorem 4.12.** *The trace, cost, and termination semantics for $\mathcal{L}$ and $\mathcal{L}^2$ coincide:*

$$\mathrm{trc}_{\mathcal{L}} = \mathrm{trc}^r_{\mathcal{L}^2}, \qquad \mathrm{cst}_{\mathcal{L}} = \mathrm{cst}^r_{\mathcal{L}^2}, \qquad \mathrm{ter}_{\mathcal{L}} = \mathrm{ter}^r_{\mathcal{L}^2}.$$

From this theorem and the compositionality of $\mathcal{L}^2$, the following is immediate:

**Theorem 4.13** (Compositionality of $\mathcal{L}$)**.** *Trace equivalence, cost equivalence and termination equivalence are congruences on the operational model of* $\mathcal{L}$*.*

The congruence of trace and termination equivalence for cool stateful SOS specifications has been shown previously by Goncharov et al. [15]. However, in contrast to the latter work, our present approach uses abstract GSOS rather than ad hoc reasoning on the cool rule format, and is based on the principled extension of $\mathcal{L}$ to $\mathcal{L}^2$. In fact, our approach provides a deeper explanation of why the cool format works: it gives rise to a lax bialgebra structure in the extended language $\mathcal{L}^2$.

## 5 Higher-Order Store

So far we have shown how to cover stateful languages with first-order store in abstract GSOS. We shall now demonstrate that the reader-writer approach to the operational semantics of stateful languages is also well-suited for more complex C-style and ML-style languages with higher-order store, where programs cannot only store basic values like integers, but also (pointers to) other programs. For that purpose, we introduce $\mathsf{Ref}^2$, a C-style, untyped imperative language with function pointers that follows the reader-writer paradigm. The presence of higher-order store yields semantics that correspond to *higher-order* abstract GSOS [16]. The inherent (bifunctorial) coalgebraic notions of program equivalence are thus fundamentally of the higher-order variety; we focus on one such notion, *termination simulation*, and employ the higher-order abstract GSOS theory to build a sound proof method for the standard *contextual equivalence* of $\mathsf{Ref}^2$.

### 5.1 An Imperative Language with Higher-Order Store

The language $\mathsf{Ref}^2$ is a basic, untyped imperative language with references and higher-order store, following mainly Reus and Streicher [36], as well as Pierce [33]. Like Reus and Streicher, we work without variable binding, e.g. $\lambda$-abstractions, allowing us to focus on the core phenomena arising from higher-order store. We outline in Section 5.5 how such additional features are incorporated.

We write *Loc* for the abstract set of *locations* and use the metavariable $l$ to denote a generic location. Locations represent regions in memory that can be allocated. The expressions of $\mathsf{Ref}^2$ are generated by the grammar below (of course, more arithmetic operations may be added at wish):

$$\mathsf{Ex} \ni e, r ::= \; l \mid n \mid !e \mid e \oplus r \mid e \ominus r \quad (l \in \mathit{Loc}, n \in \mathbb{Z}). \tag{30}$$

The set $\mathcal{S}(X)$ of *stores* in $\mathsf{Ref}^2$ is parameterized by a set $X$, which abstracts the set of readers being stored, and is defined to be the set of *partial* maps

$$\mathcal{S}(X) = \mathit{Loc} \rightharpoonup V(X) \qquad \text{where} \qquad V(X) = \mathit{Loc} + \mathbb{Z} + X.$$

Given $s \in \mathcal{S}(X)$ we write $s[l \mapsto v]$ for the store that maps $l$ to $v$ and otherwise equals $s$. The partial evaluation of expressions is given by the map $\mathsf{eev}_X \colon \mathcal{S}(X) \times \mathsf{Ex} \rightharpoonup V(X)$, natural in $X$, defined by

$$\begin{aligned}
\mathsf{eev}_X(s, l) &= l, \qquad \mathsf{eev}_X(s, n) = n, \\
\mathsf{eev}_X(s, !e) &= s(l) \quad \text{if} \quad \mathsf{eev}_X(s, e) = l, \\
\mathsf{eev}_X(s, e_1 \oplus e_2) &= n_1 + n_2 \quad \text{if} \quad \mathsf{eev}_X(s, e_i) = n_i, \\
\mathsf{eev}_X(s, e_1 \ominus e_2) &= n_1 - n_2 \quad \text{if} \quad \mathsf{eev}_X(s, e_i) = n_i.
\end{aligned}$$

According to the above, expressions may evaluate to either a location, an integer, or an element of $X$. Stores are accessed by invoking the dereferencing operator ! on an expression that evaluates to a valid location, meaning one in the domain of the store.

Analogous to the language $\mathsf{Imp}^2$, the terms of $\mathsf{Ref}^2$ are divided into two sorts, *readers* and *writers*, and are inductively generated by the following grammar:

$$\mathsf{R} \ni p, q ::= \mathsf{skip} \mid \mathsf{while}\ e\ p \mid e \coloneqq p \mid \mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q \mid p\,;q \mid \&p \mid \mathsf{expr}\ e \mid \mathsf{proc}\ p \qquad (e \in \mathsf{Ex})$$
$$\mathsf{W} \ni c ::= e \coloneqq c \mid c\,;q \mid \&c \mid s.c \mid [p]_s \mid \mathsf{ret}_{v,s} \mid \mathsf{ret}_s \qquad (s \in \mathcal{S}(\mathsf{R}), v \in V(\mathsf{R}))$$

We highlight the important added features in comparison to $\mathsf{Imp}^2$. The reader $\&p$ first evaluates $p$, then allocates a new region in memory containing the value of $p$ and returns the location of this region; proc $p$ returns the reader $p$ as a value; and expr $e$ evaluates $e$, returning the result. The behaviour of writers also changes as they no longer only return an output store, but potentially also a *value* in $V(\mathsf{R})$, that is, an integer, a location, or a reader term. Differences aside, $\mathsf{Ref}^2$ is another instance of the two-sorted reader-writer approach to stateful languages that was earlier exemplified by $\mathsf{Imp}^2$. In particular, it also features writers $[p]_s$, ret and $s.c$, and is implicitly an extension of a single-sorted language Ref whose program terms correspond to the readers of $\mathsf{Ref}^2$.

The full operational semantics of $\mathsf{Ref}^2$ are specified by the inductive rules in Figure 4. The rules apply to stores $s \in \mathcal{S}(\mathsf{R})$ and specify transitions of type $p, s \to c$ where $p \in \mathsf{R}$ and $c \in \mathsf{W}$ ('given an input store $s \in \mathcal{S}(\mathsf{R})$, the reader $p$ continues as $c$'), and transitions $c \to d \,/\, c \xrightarrow{s} d \,/\, c \downarrow s \,/\, c \downarrow v, s$ where $c, d \in \mathsf{W}$ and $v \in V(\mathsf{R})$. The first three types of writer transitions are as in $\mathsf{Imp}^2$, the last one means '$c$ terminates with value $v$ and output store $s$'. The third rule for the allocation operator $\&$ allows the choice of an *arbitrary* fresh location $l$, making writer transitions nondeterministic. Reader transitions are (partial) deterministic: there might be no transition $p, s \to c$ for given $p$ and $s$ if expression evaluation fails or gives the wrong type of value, e.g. in the rule for while.

**Example 5.1** (Landin's knot)**.** Landin's knot [27] is a programming gadget to encode general recursion into languages with higher-order store but without explicit recursion primitives. For $\mathsf{Ref}^2$, which is an untyped language, Landin's knot can be implemented via the simple pattern

$$l \coloneqq \mathsf{proc}\ p\,;\mathsf{expr}\ !l,$$

where $l$ is a location and $p$ is a reader that makes use of the expression expr $!l$. For instance, the reader $l \coloneqq \mathsf{proc}(\mathsf{expr}\ !l)\,;\mathsf{expr}\ !l$ diverges and the reader

$$l \coloneqq \mathsf{proc}(\mathsf{if}\ l'\ \mathsf{then}\ q\,; l' \coloneqq \mathsf{expr}(!l' \ominus 1)\,;\mathsf{expr}\ !l\ \mathsf{else}\ \mathsf{skip})\,; l' \coloneqq \mathsf{expr}\ 10\,;\mathsf{expr}\ !l$$

acts as an iterator. Parentheses have been added for readability.

### 5.2 Program Equivalence in $\mathsf{Ref}^2$

Our treatment of program equivalence on $\mathsf{Ref}^2$ proceeds along the lines of termination equivalence in $\mathsf{Imp}^2$ (Section 3.4), adapted to higher-order stores. Let us first recall the standard program equivalence for languages with higher-order store, which is *contextual equivalence*.

**Notation 5.2.** For $p \in \mathsf{R}$, $c, d \in \mathsf{W}$, $s \in \mathcal{S}(\mathsf{R})$ and $v \in V(\mathsf{R})$ we write

- $c \Rightarrow d$ and $c \overset{s}{\Rightarrow} d$ if there exist $n \geq 0$ and $c = c_0, \ldots, c_n = d \in \mathsf{W}$ such that, for each $i < n$, either $c_i \to c_{i+1}$ or $c_i \xrightarrow{s_{i+1}} c_{i+1}$ for some $s_{i+1} \in \mathcal{S}(\mathsf{R})$;
- $c \Downarrow v, s$ if there exists $c' \in \mathsf{W}$ such that $c \Rightarrow c' \downarrow v, s$;
- $c \Downarrow s$ if there exists $c' \in \mathsf{W}$ such that $c \Rightarrow c' \downarrow s$;
- $c \Downarrow$ if either $c \Downarrow v, s$ for some $v \in V(\mathsf{R})$ and $s \in \mathcal{S}(\mathsf{R})$, or $c \Downarrow s$ for some $s \in \mathcal{S}(\mathsf{R})$;
- $p, s \Downarrow$ if there exists $c \in \mathsf{W}$ such that $p, s \to c$ and $c \Downarrow$.

**Definition 5.3.** Let $\Sigma$ be the two-sorted signature of $\mathsf{Ref}^2$.

$$\frac{}{p\,;q, s \to [p]_s\,;q} \qquad \frac{}{\mathsf{proc}\ p, s \to \mathsf{ret}_{p,s}} \qquad \frac{}{\&p, s \to \&[p]_s}$$

$$\frac{}{\mathsf{skip}, s \to \mathsf{ret}_s} \qquad \frac{}{e \coloneqq p, s \to e \coloneqq [p]_s} \qquad \frac{\mathsf{eev}_{\mathsf{r}}(e, s) = 0}{\mathsf{while}\ e\ p, s \to \mathsf{ret}_s}$$

$$\frac{\mathsf{eev}_{\mathsf{R}}(e, s) = n \quad n \neq 0}{\mathsf{while}\ e\ p, s \to s.[p; \mathsf{while}\ e\ p]_s} \qquad \frac{\mathsf{eev}_{\mathsf{R}}(e, s) = 0}{\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q, s \to s.[q]_s}$$

$$\frac{\mathsf{eev}_{\mathsf{R}}(e, s) = n \quad n \neq 0}{\mathsf{if}\ e\ \mathsf{then}\ p\ \mathsf{else}\ q, s \to s.[p]_s} \qquad \frac{\mathsf{eev}_{\mathsf{R}}(e, s) = v \quad v \in \mathit{Loc} + \mathbb{Z}}{\mathsf{expr}\ e, s \to \mathsf{ret}_{v,s}} \qquad \frac{\mathsf{eev}_{\mathsf{R}}(e, s) = p}{\mathsf{expr}\ e, s \to s.[p]_s}$$

$$\frac{p, s \to c}{[p]_s \to c} \qquad \frac{}{\mathsf{ret}_{v,s} \downarrow v, s} \qquad \frac{}{\mathsf{ret}_s \downarrow s} \qquad \frac{}{s.c \xrightarrow{s} c}$$

$$\frac{c \xrightarrow{s} d}{\&c \xrightarrow{s} d} \qquad \frac{c \to d}{\&c \to \&d} \qquad \frac{c \downarrow v, s \quad l \notin \mathrm{dom}(s)}{\&c \downarrow l, s[l \mapsto v]}$$

$$\frac{c \xrightarrow{s} d}{c\,;q \xrightarrow{s} d\,;q} \qquad \frac{c \to d}{c\,;q \to d\,;q} \qquad \frac{c \downarrow v, s}{c\,;q \xrightarrow{s} [q]_s} \qquad \frac{c \downarrow s}{c\,;q \to [q]_s}$$

$$\frac{c \xrightarrow{s} d}{e \coloneqq c \xrightarrow{s} e \coloneqq d} \qquad \frac{c \to d}{e \coloneqq c \to e \coloneqq d} \qquad \frac{\mathsf{eev}_{\mathsf{R}}(e, s) = l \quad c \downarrow v, s}{e \coloneqq c \downarrow s[l \mapsto v]}$$

Fig. 4. Reader semantics of $\mathsf{Ref}^2$ (top) and writer semantics of $\mathsf{Ref}^2$ (bottom).

(1) A *context* is a $\Sigma$-term $C$ in a single variable '$\cdot$' (its *hole*) that occurs at most once in $C$. For $\mathsf{s}, \mathsf{t} \in \{\mathsf{r}, \mathsf{w}\}$ we write $C_{\mathsf{t}}[\cdot_{\mathsf{s}}]$ for a context $C$ of output sort t with a hole of sort s. Moreover, $C[t]$ denotes the term obtained by substituting a term $t \in \mathsf{R} \cup \mathsf{W}$ of suitable sort for the hole.

(2) *Contextual equivalence* $\equiv^{\mathrm{ctx}}$ is the two-sorted equivalence relation on $(\mathsf{R}, \mathsf{W})$ defined by

- $p \equiv^{\mathrm{ctx}}_{\mathsf{r}} q$ if (i) $\forall C_{\mathsf{r}}[\cdot_{\mathsf{r}}].\, \forall s.\, C[p], s \Downarrow \iff C[q], s \Downarrow$ and (ii) $\forall C_{\mathsf{w}}[\cdot_{\mathsf{r}}].\, C[p] \Downarrow \iff C[q] \Downarrow$;
- $c \equiv^{\mathrm{ctx}}_{\mathsf{w}} d$ if (i) $\forall C_{\mathsf{r}}[\cdot_{\mathsf{w}}].\forall s.\, C[c], s \Downarrow \iff C[d], s \Downarrow$ and (ii) $\forall C_{\mathsf{w}}[\cdot_{\mathsf{w}}].\, C[c] \Downarrow \iff C[d] \Downarrow$.

A standard observation is that contextual equivalence can be characterized alternatively as the greatest congruence relation on the set of programs that is adequate with respect to termination:

**Definition 5.4.** (1) A ($\mathsf{Ref}^2$)-*congruence* is a congruence on the initial algebra $(\mathsf{R}, \mathsf{W})$ for $\Sigma$, that is, a two-sorted relation $R = (R_{\mathsf{r}} \subseteq \mathsf{R} \times \mathsf{R}, R_{\mathsf{w}} \subseteq \mathsf{W} \times \mathsf{W})$ compatible with all constructors of $\mathsf{Ref}^2$.

(2) A two-sorted relation $R = (R_{\mathsf{r}} \subseteq \mathsf{R} \times \mathsf{R}, R_{\mathsf{w}} \subseteq \mathsf{W} \times \mathsf{W})$ is (*termination-*)*adequate* if

(a) for all $p, q \in \mathsf{R}$ and $s \in \mathcal{S}(\mathsf{R})$, if $R_{\mathsf{r}}(p, q)$ then $p, s \Downarrow \iff q, s \Downarrow$;

(b) for all $c, d \in \mathsf{W}$, if $R_{\mathsf{w}}(c, d)$ then $c \Downarrow \iff d \Downarrow$.

**Proposition 5.5.** *Contextual equivalence is the greatest adequate congruence.*

It is worth mentioning that mere termination equivalence is no longer a suitable program equivalence in $\mathsf{Ref}^2$, as it only relates programs that return strictly equal stores. This is too restrictive in the higher-order setting: the returned stores may contain readers that should themselves be related, but not necessarily be syntactically equal. However, to reason about contextual equivalence

in an efficient manner, we use the same method as for $\mathsf{Imp}^2$, namely that of *termination simulations*, which is a higher-order variant of the notion introduced in Definition 3.9.

**Notation 5.6.** For every relation $R \subseteq X \times X$, we define the relation $V(R) \subseteq V(X) \times V(X)$ by

$$V(R) = \Delta_{Loc} \cup \Delta_{\mathbb{Z}} \cup R,$$

and the relation $\mathcal{S}(R) \subseteq \mathcal{S}(X) \times \mathcal{S}(X)$ by

$$\mathcal{S}(R) = \{(s_1, s_2) \mid \mathrm{dom}(s_1) = \mathrm{dom}(s_2) \ \wedge\ \forall l \in \mathrm{dom}(s_1).\, V(R)(s_1(l), s_2(l))\}.$$

**Definition 5.7.** A two-sorted relation $R = (R_{\mathsf{r}} \subseteq \mathsf{R}\times\mathsf{R}, R_{\mathsf{w}} \subseteq \mathsf{W}\times\mathsf{W})$ is a (*higher-order*) *termination simulation* if the following conditions hold for all $p, q \in \mathsf{R}$, $c, d \in \mathsf{W}$, $s \in \mathcal{S}(\mathsf{R})$ and $v \in V(\mathsf{R})$:

(1) If $R_{\mathsf{r}}(p, q)$ and $p, s \to c$ then there exists $d$ such that $q, s \to d$ and $R_{\mathsf{w}}(c, d)$.

(2) If $R_{\mathsf{w}}(c, d)$ and $c \xrightarrow{s} c'$ or $c \to c'$, then there exists $d'$ such that $d \Rightarrow d'$ and $R_{\mathsf{w}}(c', d')$.

(3) If $R_{\mathsf{w}}(c, d)$ and $c \downarrow s$, then there exists $s'$ such that $d \Downarrow s'$ and $\mathcal{S}(R_{\mathsf{r}})(s, s')$.

(4) If $R_{\mathsf{w}}(c, d)$ and $c \downarrow v, s$, then there exist $v', s'$ such that $d \Downarrow v', s'$ and $V(R_{\mathsf{r}})(v, v')$ and $\mathcal{S}(R_{\mathsf{r}})(s, s')$.

*Termination similarity*, denoted by $\preceq = (\preceq_{\mathsf{r}}, \preceq_{\mathsf{w}})$, is the greatest termination simulation.

We denote the symmetrization of termination similarity by $\equiv = (\equiv_{\mathsf{r}}, \equiv_{\mathsf{w}}) = (\preceq_{\mathsf{r}} \cap \succeq_{\mathsf{r}}, \preceq_{\mathsf{w}} \cap \succeq_{\mathsf{w}})$, where $\succeq_{\mathsf{r/w}}$ is the converse of $\preceq_{\mathsf{r/w}}$. From the definition of similarity, we immediately obtain:

**Proposition 5.8** (Adequacy)**.** *The relation $\equiv$ is adequate.*

**Remark 5.9.** The stricter notion of termination equivalence is recovered by requiring $d \Downarrow s$ and $d \Downarrow v, s$ in clauses (3) and (4) of Definition 5.7. Termination equivalence is included in $\equiv$.

We give a few examples of termination-similar terms.

**Example 5.10** (Skip-ing)**.** For any reader $p \in \mathsf{R}$, one has $\mathsf{skip}\,; p \equiv_{\mathsf{r}} p$ because the two programs are termination equivalent. The relation $(T_{\mathsf{r}}, T_{\mathsf{w}})$ below and its converse are termination simulations:

$$T_{\mathsf{r}} = \ \Delta_{\mathsf{r}} \cup \{(\mathsf{skip}\,; p, p)\}, \qquad T_{\mathsf{w}} = \ \Delta_{\mathsf{w}} \cup \bigcup\nolimits_{s \in \mathcal{S}(\mathsf{R})} \{([\mathsf{skip}]_s\,; p, c_s), (\mathsf{ret}_s\,; p, c_s), ([p]_s, c_s)\},$$

**Example 5.11.** For any location $l$, one has $l \coloneqq 2\,; l \coloneqq \mathsf{expr}(!l \oplus 2) \quad \equiv_{\mathsf{r}} \quad l \coloneqq 2\,; l \coloneqq \mathsf{expr}(!l \oplus !l)$.

**Example 5.12.** Given a termination simulation $(T_{\mathsf{r}}, T_{\mathsf{w}})$ and a location $l$, one can build a new termination simulation $(Q_{\mathsf{r}}, Q_{\mathsf{w}})$ by setting

$$Q_{\mathsf{r}} = T_{\mathsf{r}} \cup \{(l \coloneqq \mathsf{proc}\ p, l \coloneqq \mathsf{proc}\ q) \mid (p, q) \in T_{\mathsf{r}}\},$$
$$Q_{\mathsf{w}} = T_{\mathsf{w}} \cup \bigcup\nolimits_{s \in \mathcal{S}(\mathsf{R}), (p,q) \in T_{\mathsf{r}}} \{(l \coloneqq [\mathsf{proc}\ p]_s, l \coloneqq [\mathsf{proc}\ q]_s), (l \coloneqq \mathsf{ret}_{p,s}, l \coloneqq \mathsf{ret}_{q,s})\}.$$

Hence, given any pair $(p, q) \in T_{\mathsf{r}}$ of related readers, $l \coloneqq \mathsf{proc}\ p \preceq_{\mathsf{r}} l \coloneqq \mathsf{proc}\ q$.

**Example 5.13** (Landin's knot)**.** Recall Landin's knot from Example 5.1. We can see that

$$l \coloneqq \mathsf{proc}(\mathsf{expr}\ !l); \mathsf{expr}\ !l \quad \equiv_{\mathsf{r}} \quad l \coloneqq \mathsf{proc}(\mathsf{while}\ 1\ \mathsf{skip}); \mathsf{expr}\ !l$$

because both programs diverge on every input state.

The following theorem is our main technical result on $\mathsf{Ref}^2$, and is proved in Sections 5.3 and 5.4.

**Theorem 5.14** (Compositionality)**.** *Termination similarity, hence also $\equiv$, forms a congruence.*

As an important corollary of Proposition 5.5, Proposition 5.8 and Theorem 5.14, we have thus established two-way termination similarity as a sound proof method for contextual equivalence:

**Corollary 5.15** (Soundness). *If $p \equiv_r q$ then $p \equiv_r^{\mathrm{ctx}} q$, and if $c \equiv_w d$ then $c \equiv_w^{\mathrm{ctx}} d$.*

The presence of a higher-order store complicates the proof of Theorem 5.14, to the extent that a standard, language-specific approach would likely require the development of a tailor-made Kripke/step-indexed logical relation or a bisimulation-based method as in [25], tasks that are both technically challenging and laborious. We will instead systematically derive the theorem using the abstract theory of congruence provided by (higher-order) abstract GSOS.

### 5.3 Categorical Modelling of Ref$^2$

As for Imp$^2$, the syntax and operational semantics Ref$^2$ are modelled over the category $\mathbf{Set}^2$. The syntax is given by the polynomial endofunctor $\Sigma$ on $\mathbf{Set}^2$ corresponding to the signature of Ref$^2$:

$$\Sigma_{\mathsf{r}}(X) = \underbrace{1}_{\text{skip}} + \underbrace{\mathsf{Ex} \times X_{\mathsf{r}}}_{:=} + \underbrace{\mathsf{Ex} \times X_{\mathsf{r}}}_{\text{while}} + \underbrace{X_{\mathsf{r}} \times X_{\mathsf{r}}}_{-;-} + \underbrace{\mathsf{Ex} \times X_{\mathsf{r}} \times X_{\mathsf{r}}}_{\text{if-statement}} + \underbrace{\mathsf{Ex}}_{\text{expr}} + \underbrace{X_{\mathsf{r}}}_{\&} + \underbrace{X_{\mathsf{r}}}_{\text{proc}},$$

$$\Sigma_{\mathsf{w}}(X) = \underbrace{X_{\mathsf{r}} \times \mathcal{S}(X_{\mathsf{r}})}_{[-]_-} + \underbrace{\mathsf{Ex} \times X_{\mathsf{w}}}_{:=} + \underbrace{X_{\mathsf{w}} \times X_{\mathsf{r}}}_{-;-} + \underbrace{X_{\mathsf{w}}}_{\&} + \underbrace{\mathcal{S}(X_{\mathsf{r}})}_{\text{ret}} + \underbrace{V(X_{\mathsf{r}}) \times \mathcal{S}(X_{\mathsf{r}})}_{\text{ret}_{-,-}} + \underbrace{\mathcal{S}(X_{\mathsf{r}}) \times X_{\mathsf{w}}}_{-.-}.$$

The initial algebra for $\mu\Sigma$ is formed by the two-sorted set $(\mathsf{R}, \mathsf{W})$ of readers and writers.

Recall that the dynamics of Imp$^2$ corresponds to a coalgebra (13). Here, from a coalgebraic standpoint, the presence of a higher-order store turns the dynamics of Ref$^2$ into a *higher-order* coalgebra [16], that is, a morphism of type $C \to B(C, C)$ where $B \colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$ is a *bifunctor* of mixed variance. Specifically, the dynamic behaviour of Ref$^2$ is modelled by the bifunctor $B \colon (\mathbf{Set}^2)^{\mathrm{op}} \times \mathbf{Set}^2 \to \mathbf{Set}^2$ given as follows:

$$B_{\mathsf{r}}(X, Y) = (Y_{\mathsf{w}} + 1)^{\mathcal{S}(X_{\mathsf{r}})} \quad \text{and} \quad B_{\mathsf{w}}(X, Y) = \mathcal{P}((V(Y_{\mathsf{r}}) + 1) \times \mathcal{S}(Y_{\mathsf{r}}) + Y_{\mathsf{w}} \times (\mathcal{S}(Y_{\mathsf{r}}) + 1)). \tag{31}$$

We regard $B$ as ordered by equality in the reader sort and by pointwise inclusion in the writer sort. The transitions of Figure 4 then induce a higher-order $B$-coalgebra

$$\gamma \colon (\mathsf{R}, \mathsf{W}) \to ((\mathsf{W} + 1)^{\mathcal{S}(\mathsf{R})}, \mathcal{P}((V(\mathsf{R}) + 1) \times \mathcal{S}(\mathsf{R}) + \mathsf{W} \times (\mathcal{S}(\mathsf{R}) + 1))) = B((\mathsf{R}, \mathsf{W}), (\mathsf{R}, \mathsf{W})) \tag{32}$$

where the reader component is given by $\gamma^{\mathsf{r}}(p)(s) = c$ if $p, s \to c$ and $\gamma^{\mathsf{r}}(p)(s) = *$ if no such transition exists, and the writer component is defined as follows for $d \in \mathsf{W}$, $s \in \mathcal{S}(\mathsf{R})$ and $v \in V(\mathsf{R})$,

$$(d, *) \;/\; (d, s) \;/\; (*, s) \;/\; (v, s) \in \gamma^{\mathsf{w}}(c) \quad \Longleftrightarrow \quad c \to d \;/\; c \xrightarrow{s} d \;/\; c \downarrow s \;/\; c \downarrow v, s.$$

The operational rules of Ref$^2$ can be modelled using a higher-order version of the notion of GSOS law. A (*0-pointed*) *higher-order GSOS law* [16] of $\Sigma \colon \mathbb{C} \to \mathbb{C}$ over $B \colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$ is a family

$$\varrho_{X,Y} \colon \Sigma(X \times B(X, Y)) \to B(X, \Sigma^\star(X + Y)) \qquad (X, Y \in \mathbb{C}) \tag{33}$$

of morphisms in $\mathbb{C}$ dinatural in $X \in \mathbb{C}$ and natural in $Y \in \mathbb{C}$. For Ref$^2$ we consider the higher-order GSOS law (33) of the syntax functor $\Sigma$ of the language over the behaviour functor (31) that, as in the first-order case, encodes the rules of the language. For instance, for each $X \in \mathbf{Set}^2$ the map

$$\varrho^{\mathsf{w}}_{X,Y} \colon \Sigma_{\mathsf{w}}(X \times B(X, Y)) \to \mathcal{P}((V(\Sigma^\star_{\mathsf{r}}(X + Y)) + 1) \times \mathcal{S}(\Sigma^\star_{\mathsf{r}}(X + Y)) + \Sigma^\star_{\mathsf{w}}(X + Y) \times (\mathcal{S}(\Sigma^\star_{\mathsf{r}}(X + Y)) + 1))$$

encodes the behaviour of the allocation operator $\& \colon \mathsf{w} \to \mathsf{w}$ as follows for $c \in X_{\mathsf{w}}$, $U \in B_{\mathsf{w}}(X, Y)$:

$$\varrho^{\mathsf{w}}_{X,Y}(\&(c, U)) = \{(\&d, s) \mid (d, s) \in U\} \cup \{\&d \mid d \in U\} \cup \{(l, s[l \mapsto v]) \mid (v, s) \in U \wedge l \notin \mathrm{dom}(s)\}.$$

See [17, App. A] for the full definition of the law for Ref$^2$. The (di)naturality of the higher-order GSOS law $\varrho$ results from the operational semantics being parametrically polymorphic w.r.t. the choices of $X$ and $Y$. In particular, it is polymorphic on the set of readers $X_{\mathsf{r}}$ in $\mathcal{S}(X_{\mathsf{r}})$.

The definition of the operational model for first-order GSOS laws extends to the present higher-order setting. Specifically, the *operational model* of $\varrho$ (33) is the higher-order coalgebra $\gamma \colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$ whose structure is the unique morphism making the diagram below commute:

$$\begin{array}{ccccc} \Sigma(\mu\Sigma) & & \xrightarrow{\quad\iota\quad} & & \mu\Sigma \\ {\scriptstyle \Sigma\langle \mathrm{id}, \gamma\rangle}\downarrow & & & & \downarrow{\scriptstyle \gamma} \\ \Sigma(\mu\Sigma \times B(\mu\Sigma, \mu\Sigma)) & \xrightarrow{\varrho_{\mu\Sigma,\mu\Sigma}} & B(\mu\Sigma, \Sigma^\star(\mu\Sigma + \mu\Sigma)) & \xrightarrow{B(\mathrm{id}, \hat{\iota}\cdot\Sigma^\star\nabla)} & B(\mu\Sigma, \mu\Sigma) \end{array} \tag{34}$$

In the case of $\mathsf{Ref}^2$ this yields precisely the higher-order coalgebra (32) that runs program terms.

### 5.4 Program Equivalence in $\mathsf{Ref}^2$ via Higher-Order Abstract GSOS

Similar to first-order abstract GSOS, a general theory of congruence of coalgebraic similarity is available in higher-order abstract GSOS. We adapt the notion of AOS (Definition 2.17) in the base category $\mathbf{Set}^T$ (for a set $T$) as follows:

**Definition 5.16.** A *higher abstract operational setting* (*HAOS*) $\mathcal{O} = (\Sigma, \overline{\Sigma}, B, \bar{B}, \varrho, \gamma, \tilde{\gamma})$ is given by

- a polynomial functor $\Sigma\colon \mathbf{Set}^T \to \mathbf{Set}^T$ with its canonical relation lifting $\overline{\Sigma}$;
- an ordered bifunctor $(B, \preceq)\colon (\mathbf{Set}^T)^{\mathrm{op}} \times \mathbf{Set}^T \to \mathbf{Set}^T$ with a relation lifting $\bar{B}$;
- a higher-order GSOS law $\varrho$ of $\Sigma$ over $B$;
- the operational model $(\mu\Sigma, \gamma)$ of $\varrho$ with a weakening $\tilde{\gamma}$.

Here, a *weakening* $\tilde{\gamma}\colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$ of the higher-order coalgebra $\gamma$ is meant to be one with respect to the endofunctor $B(\mu\Sigma, -)\colon \mathbf{Set}^T \to \mathbf{Set}^T$ and its lifting $\bar{B}(\Delta_{\mu\Sigma}, -)\colon \mathbf{Rel}(\mathbf{Set}^T) \to \mathbf{Rel}(\mathbf{Set}^T)$ as in Definition 2.12. The compositionality theorem for first-order abstract GSOS (Theorem 2.20) then generalizes accordingly, and is another special case of [43, Cor. VIII.7]. Its proof in *op. cit.* is based on a categorical abstraction of *Howe's method* [20, 21], a standard technique for establishing congruence of (bi)similarity for higher-order languages.

**Theorem 5.17** (Compositionality)**.** *Suppose that* $\mathcal{O} = (\Sigma, \overline{\Sigma}, B, \bar{B}, \varrho, \gamma, \tilde{\gamma})$ *is a HAOS such that*

(1) $\bar{B}$ *is up-closed and satisfies* $\Delta_{B(X,Y)} \subseteq \bar{B}(\Delta_X, \Delta_Y)$ *for all* $X, Y \in \mathbf{Set}^T$ *and*

$$\bar{B}(R, S) \cdot \bar{B}(\Delta_X, T) \subseteq \bar{B}(R, S \bullet T) \quad \text{for all } R \rightarrowtail X \times X \text{ and } S, T \rightarrowtail Y \times Y;$$

(2) $\varrho$ *is liftable;*

(3) $(\mu\Sigma, \iota, \tilde{\gamma})$ *is a higher-order lax* $\varrho$*-bialgebra.*

*Then the* $\bar{B}(\Delta_{\mu\Sigma}, -)$*-similarity relation on* $(\mu\Sigma, \gamma, \tilde{\gamma})$ *is a* $\overline{\Sigma}$*-congruence on the initial algebra* $(\mu\Sigma, \iota)$.

Here (2) means that for each $R \rightarrowtail X \times X$ and $S \rightarrowtail Y \times Y$ the map $\varrho_{X,Y}$ is a relation morphism from $\overline{\Sigma}(R \times \bar{B}(R, S))$ to $\bar{B}(R, \overline{\Sigma^\star}(R + S))$, and (3) means that the diagram (34) commutes laxly when $\gamma$ is replaced with $\tilde{\gamma}$, that is, $B(\mathrm{id}, \hat{\iota} \cdot \Sigma^\star\nabla) \cdot \varrho_{\mu\Sigma,\mu\Sigma} \cdot \Sigma\langle \mathrm{id}, \tilde{\gamma}\rangle \preceq \tilde{\gamma} \cdot \iota$.

The compositionality of termination similarity in $\mathsf{Ref}^2$ (Theorem 5.14) is an instance of the compositionality result for higher-order abstract GSOS: we apply Theorem 5.17 to $\mathcal{O} = (\Sigma, \overline{\Sigma}, B, \bar{B}, \varrho, \gamma, \tilde{\gamma})$ where $\varrho$ is the higher-order GSOS law for $\mathsf{Ref}^2$, the relation lifting $\bar{B}$ of $B$ (31) is defined by

$$(\bar{B}(R, S))_{\mathsf{r}} = \{(f_1, g_1) \mid \forall s_1, s_2.\, \mathcal{S}(R_{\mathsf{r}})(s_1, s_2) \implies f_1(s_1) = f_2(s_2) = \star \vee S_{\mathsf{w}}(f_1(s_1), f_2(s_2))\},$$

$$(\bar{B}(R, S))_{\mathsf{w}} = \vec{\mathcal{P}}((V(S_{\mathsf{r}}) + \Delta_1) \times \mathcal{S}(S_{\mathsf{r}}) + S_{\mathsf{w}} \times (\mathcal{S}(S_{\mathsf{r}}) + \Delta_1)),$$

and $\tilde{\gamma}\colon (\mathsf{R}, \mathsf{W}) \to B((\mathsf{R}, \mathsf{W}), (\mathsf{R}, \mathsf{W}))$ is the weakening of $\gamma$ given by Notation 5.2:

$$\tilde{\gamma}^{\mathsf{r}} = \gamma^{\mathsf{r}} \quad \text{and} \quad \tilde{\gamma}^{\mathsf{w}}(c) = \{d \mid c \Rightarrow d\} \cup \{(d, s) \mid c \Rightarrow d, s\} \cup \{s \mid d \Downarrow s\} \cup \{(v, s) \mid c \Downarrow v, s\}.$$

Observe that $\bar{B}(\Delta_{\mu\Sigma}, -)$-similarity on the operational model $((\mathsf{R}, \mathsf{W}), \gamma, \tilde{\gamma})$ is termination similarity (Definition 5.7). It is then a matter of routine calculations, following the definitions of the ingredients of $\mathcal{O}$, to verify that the conditions (1)–(3) of Theorem 5.17 hold. Once again, (2) boils down to parametric polymorphism of the rules of $\mathsf{Ref}^2$, and (3) to the rules being sound for weak transitions.

### 5.5 Towards Variable Binding and Higher-Order Features

The reader-writer approach to the operational semantics of stateful languages also applies to higher-order languages that feature higher-order functions, variable binding and substitution. More so, at first glance, this combination of reader-writer semantics with higher-order features seems to be compatible with higher-order abstract GSOS (as its constituents demonstrably are). We shall now briefly sketch the key ideas of a reader-writer, call-by-name, untyped $\lambda$-calculus with higher-order store (which we call $\lambda$-Ref$^2$) in higher-order abstract GSOS, by extending Ref$^2$ accordingly.

The *reader* syntax of $\lambda$-Ref$^2$ extends that of Ref$^2$ by adding variables $x$, $\lambda$-abstractions $\lambda x.p$ and applications $p\ q$. On the side of *writers*, we add "ongoing" applications, i.e. expressions of the form $c\ q$. The operational semantics of the new constructs are as follows:

$$\frac{}{p\ q, s \to [p]_s\ q} \qquad \frac{c \to d}{c\ q \to d\ q} \qquad \frac{c \xrightarrow{s} d}{c\ q \xrightarrow{s} d\ q} \qquad \frac{}{[\lambda x.p]_s\ q \xrightarrow{s} [p[q/x]]_s}$$

The operational semantics of $\lambda$-Ref$^2$ live in $(\mathbf{Set}^{\mathbb{F}})^2$, the category of two-sorted (covariant) presheaves over the category $\mathbb{F}$ of finite cardinals and functions (representing untyped variable contexts). Modelling syntax for languages with variable binding in such presheaf categories is standard, following Fiore et al. [14]. In this new setting, an object $X \in (\mathbf{Set}^{\mathbb{F}})^2$ is a pair of presheaves $(X_\mathrm{r}, X_\mathrm{w})$ in $\mathbf{Set}^{\mathbb{F}}$. Specifically for $\lambda$-Ref$^2$, the object of terms would be $(\mathsf{R}, \mathsf{W})$ where $\mathsf{R}(n)$ and $\mathsf{W}(n)$ are the sets of readers and writers in the variable context $x_1, \ldots, x_n$.

Implementing $\lambda$-Ref$^2$ as a higher-order GSOS law requires modelling $\lambda$-abstractions as functions on readers. The behaviour corresponding to $\lambda$-abstractions should thus be given by an exponential $Y_\mathrm{r}^{X_\mathrm{r}}$ (We omit some technical details of the behaviour, such as a reader-reader substitution structure of terms [16]). The semantics of $\lambda$-abstractions is given by a reader-labelled transition system, e.g.

$$\frac{}{\lambda x.p \xrightarrow{q} p[q/x]} \qquad \frac{p \xrightarrow{r} q}{[p]_s \xrightarrow{r} [q]_s, s} \qquad \frac{c \xrightarrow{q,s} p'}{c\ q \xrightarrow{s} p'}.$$

In the above, $\lambda x.p$ behaves as a function mapping a reader $q$ to $p[x/q]$. Under this perspective, which is standard in higher-order abstract GSOS [16, §5], $\beta$-reduction in $\lambda$-Ref$^2$ is implemented in terms of the two rules for resp. $[-]_s$ and application. The exact details of the above semantics, as well as the development of a theory of compositionality for $\lambda$-Ref$^2$ and similar languages via the methods provided by higher-order abstract GSOS, are left as a prospect for future work.

## 6 Conclusion and Future Work

We have presented a systematic approach to the operational semantics of stateful languages, based on the formal distinction between readers and writers. This approach is capable of tapping into the powerful theory of (higher-order) abstract GSOS, refuting the accepted surmise that stateful languages are not compatible with the framework. Taking advantage of this fact, we derive efficient reasoning techniques for program equivalence for both first-order and higher-order store. The extension to fully fledged higher-order languages as outlined in Section 5.5 is a natural next step.

Our approach notably does not treat the semantics of state in terms of the *state monad*, as is often the case in works about program equivalence on the $\lambda$-calculus with algebraic effects [13, 23, 35]. Even though the distinction between readers and writers alludes to the two components of the state monad, it is currently unclear if there is some formal connection to the latter. However, it is worth pointing out that the divergence of Abou-Saleh and Pattinson [2, 3] from the Turi-Plotkin framework can be partly attributed to the non-commutativity of the state monad. Now, with a satisfactory abstract GSOS based approach to state, revisiting abstract GSOS in Kleisli categories of *commutative* monads, e.g. the power set monad for trace semantics, has become a possibility.

## Acknowledgments

Sergey Goncharov and Stelios Tsampas acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project numbers 527481841. Stefan Milius was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 517924115. Lutz Schröder was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 531706730. Henning Urbat is supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 470467389.

## References

[1] Andreas Abel, Guillaume Allais, Aliya Hameer, Brigitte Pientka, Alberto Momigliano, Steven Schäfer, and Kathrin Stark. 2019. POPLMark reloaded: Mechanizing proofs by logical relations. *J. Funct. Program.* 29 (2019), e19. https://doi.org/10.1017/S0956796819000170

[2] Faris Abou-Saleh and Dirk Pattinson. 2011. Towards Effects in Mathematical Operational Semantics. In *Mathematical Foundations of Programming Semantics, MFPS 2011 (ENTCS, Vol. 276)*, Michael W. Mislove and Joël Ouaknine (Eds.). Elsevier, 81–104. https://doi.org/10.1016/j.entcs.2011.09.016

[3] Faris Abou-Saleh and Dirk Pattinson. 2013. Comodels and Effects in Mathematical Operational Semantics. In *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013 (Lecture Notes in Computer Science, Vol. 7794)*, Frank Pfenning (Ed.). Springer, 129–144. https://doi.org/10.1007/978-3-642-37075-5_9

[4] Danel Ahman, Cédric Fournet, Cătălin Hriţcu, Kenji Maillard, Aseem Rastogi, and Nikhil Swamy. 2017. Recalling a witness: foundations and applications of monotonic state. *Proc. ACM Program. Lang.* 2, POPL, Article 65 (Dec. 2017), 30 pages. https://doi.org/10.1145/3158153

[5] Amal Ahmed, Nick Benton, Lars Birkedal, and Martin Hofmann (Eds.). 2010. *Modelling, Controlling and Reasoning About State, 29.08. - 03.09.2010.* Dagstuhl Seminar Proceedings, Vol. 10351. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany. http://drops.dagstuhl.de/portals/10351/

[6] Amal Ahmed, Nick Benton, Martin Hofmann, and Greg Morrisett (Eds.). 2008. *Types, Logics and Semantics for State, 03.02. - 08.02.2008.* Dagstuhl Seminar Proceedings, Vol. 08061. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. http://drops.dagstuhl.de/portals/08061/

[7] Amal Jamil Ahmed. 2004. *Semantics of types for mutable state.* Ph. D. Dissertation. USA. AAI3136691.

[8] Michael Barr. 1970. Coequalizers and free triples. *Math. Z.* 116 (1970), 307–322. https://doi.org/10.1007/BF01111838

[9] Nick Benton and Chung-Kil Hur. 2009. Biorthogonality, Step-Indexing and Compiler Correctness. In *14th ACM SIGPLAN International Conference on Functional Programming (ICFP 2009)*. ACM, 97–108. https://doi.org/10.1145/1596550.1596567

[10] Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang. 2011. Step-indexed kripke models over recursive worlds. *SIGPLAN Not.* 46, 1 (Jan. 2011), 119–132. https://doi.org/10.1145/1925844.1926401

[11] Bard Bloom. 1995. Structural Operational Semantics for Weak Bisimulations. *Theor. Comput. Sci.* 146, 1&2 (1995), 25–68. https://doi.org/10.1016/0304-3975(94)00152-9

[12] Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. 2015. Lax Bialgebras and Up-To Techniques for Weak Bisimulations. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015.* 240–253. https://doi.org/10.4230/LIPIcs.CONCUR.2015.240

[13] Ugo Dal Lago, Francesco Gavazzo, and Paul Blain Levy. 2017. Effectful applicative bisimilarity: Monads, relators, and Howe's method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005117

[14] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999)*. IEEE Computer Society, 193–202. https://doi.org/10.1109/LICS.1999.782615

[15] Sergey Goncharov, Stefan Milius, Lutz Schröder, Stelios Tsampas, and Henning Urbat. 2022. Stateful Structural Operational Semantics. In *7th International Conference on Formal Structures for Computation and Deduction, FSCD'22 (LIPIcs, Vol. 228)*, Amy P. Felty (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:19. https://doi.org/10.4230/LIPIcs.FSCD.2022.30

[16] Sergey Goncharov, Stefan Milius, Lutz Schröder, Stelios Tsampas, and Henning Urbat. 2023. Towards a Higher-Order Mathematical Operational Semantics. In *50th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2023) (Proc. ACM Program. Lang., Vol. 7)*. ACM, Article 22, 27 pages. https://doi.org/10.1145/3571215

[17] Sergey Goncharov, Stefan Milius, Lutz Schröder, Stelios Tsampas, and Henning Urbat. 2025. Bialgebraic Reasoning on Stateful Languages. *CoRR* abs/2503.10955 (2025). https://doi.org/10.48550/ARXIV.2503.10955 arXiv:2503.10955

[18] Robert Harper. 1994. A simplified account of polymorphic references. *Inform. Process. Lett.* 51, 4 (1994), 201–206. https://doi.org/10.1016/0020-0190(94)90120-1

[19] Claudio Hermida and Bart Jacobs. 1998. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation* 145, 2 (1998), 107–152. https://doi.org/10.1006/inco.1998.2725

[20] Douglas J. Howe. 1989. Equality In Lazy Computation Systems. In *4th Annual Symposium on Logic in Computer Science (LICS 1989)*. IEEE Computer Society, 198–203. https://doi.org/10.1109/LICS.1989.39174

[21] Douglas J. Howe. 1996. Proving Congruence of Bisimulation in Functional Programming Languages. *Inf. Comput.* 124, 2 (1996), 103–112. https://doi.org/10.1006/inco.1996.0008

[22] Bart Jacobs. 2016. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge Tracts in Theoretical Computer Science, Vol. 59. Cambridge University Press. https://doi.org/10.1017/CBO9781316823187

[23] Patricia Johann, Alex Simpson, and Janis Voigtländer. 2010. A Generic Operational Metatheory for Algebraic Effects. In *25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010)*. IEEE Computer Society, 209–218. https://doi.org/10.1109/LICS.2010.29

[24] Ralf Jung, Robbert Krebbers, Lars Birkedal, and Derek Dreyer. 2016. Higher-order ghost state. *SIGPLAN Not.* 51, 9 (Sept. 2016), 256–269. https://doi.org/10.1145/3022670.2951943

[25] Vasileios Koutavas and Mitchell Wand. 2006. Small bisimulations for reasoning about higher-order imperative programs. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006*, J. Gregory Morrisett and Simon L. Peyton Jones (Eds.). ACM, 141–152. https://doi.org/10.1145/1111037.1111050

[26] Alexander Kurz and Jiří Velebil. 2016. Relation Lifting, a Survey. *Journal of Logical and Algebraic Methods in Programming* 85, 4 (2016), 475–499. https://doi.org/10.1016/j.jlamp.2015.08.002

[27] P. J. Landin. 1964. The Mechanical Evaluation of Expressions. *Comput. J.* 6, 4 (1964), 308–320. https://doi.org/10.1093/comjnl/6.4.308

[28] Paul Levy, John Power, and Hayo Thielecke. 2003. Modelling environments in call-by-value programming languages. *Inf. Comput.* 185, 2 (2003), 182–210. https://doi.org/10.1016/S0890-5401(03)00088-9

[29] Paul Blain Levy. 2001. *Call-by-push-value*. Ph. D. Dissertation. Queen Mary University of London, UK. https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.369233

[30] S. Mac Lane. 1978. *Categories for the Working Mathematician* (2 ed.). Graduate Texts in Mathematics, Vol. 5. Springer. http://link.springer.com/10.1007/978-1-4757-4721-8

[31] Ian Mason and Carolyn Talcott. 1991. Equivalence in functional languages with effects. *Journal of Functional Programming* 1, 3 (1991), 287–327. https://doi.org/10.1017/S0956796800000125

[32] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local Reasoning about Programs that Alter Data Structures. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL '01)*. Springer-Verlag, Berlin, Heidelberg, 1–19.

[33] Benjamin C. Pierce. 2002. *Types and programming languages*. MIT Press.

[34] Andrew M. Pitts and Ian D. B. Stark. 1998. Operational Reasoning for Functions with Local State. In *Higher Order Operational Techniques in Semantics*, Andrew D. Gordon and Andrew M. Pitts (Eds.). Cambridge University Press, New York, NY, USA, 227–274.

[35] Gordon D. Plotkin and John Power. 2001. Adequacy for Algebraic Effects. In *Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001 (Lecture Notes in Computer Science, Vol. 2030)*, Furio Honsell and Marino Miculan (Eds.). Springer, 1–24. https://doi.org/10.1007/3-540-45315-6_1

[36] Bernhard Reus and Thomas Streicher. 2005. About Hoare Logics for Higher-Order Store. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005 (Lecture Notes in Computer Science, Vol. 3580)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer, 1337–1348. https://doi.org/10.1007/11523468_108

[37] John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS '02)*. IEEE Computer Society, USA, 55–74. https://doi.org/10.1109/LICS.2002.1029817

[38] Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. 2022. Denotational semantics of general store and polymorphism. *CoRR* abs/2210.02169 (2022). https://doi.org/10.48550/ARXIV.2210.02169 arXiv:2210.02169

[39] Robert D. Tennent and Dan R. Ghica. 2000. Abstract Models of Storage. *High. Order Symb. Comput.* 13, 1/2 (2000), 119–129. https://doi.org/10.1023/A:1010022312623

[40] Amin Timany, Léo Stefanesco, Morten Krogh-Jespersen, and Lars Birkedal. 2018. A logical relation for monadic encapsulation of state: proving contextual equivalences in the presence of runST. *Proc. ACM Program. Lang.* 2, POPL (2018), 64:1–64:28. https://doi.org/10.1145/3158152

[41] Daniele Turi and Gordon D. Plotkin. 1997. Towards a Mathematical Operational Semantics. In *12th Annual IEEE Symposium on Logic in Computer Science (LICS 1997)*. 280–291. https://doi.org/10.1109/LICS.1997.614955

[42] Aaron Joseph Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. 2013. Logical relations for fine-grained concurrency. In *40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '13)*, Roberto Giacobazzi and Radhia Cousot (Eds.). ACM, 343–356. https://doi.org/10.1145/2429069.2429111

[43] Henning Urbat, Stelios Tsampas, Sergey Goncharov, Stefan Milius, and Lutz Schröder. 2023. Weak Similarity in Higher-Order Mathematical Operational Semantics. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*. IEEE, 1–13. https://doi.org/10.1109/LICS56636.2023.10175706

[44] Henning Urbat, Stelios Tsampas, Sergey Goncharov, Stefan Milius, and Lutz Schröder. 2023. Weak Similarity in Higher-Order Mathematical Operational Semantics. arXiv:2302.08200 [cs.PL]

[45] Viktor Vafeiadis. 2011. Concurrent Separation Logic and Operational Semantics. *Electronic Notes in Theoretical Computer Science* 276 (2011), 335–351. https://doi.org/10.1016/j.entcs.2011.09.029 Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVII).

[46] Rob J. van Glabbeek. 2011. On cool congruence formats for weak bisimulations. *Theor. Comput. Sci.* 412, 28 (2011), 3283–3302. https://doi.org/10.1016/j.tcs.2011.02.036

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

[48] Nobuko Yoshida, Kohei Honda, and Martin Berger. 2008. Logical Reasoning for Higher-Order Functions with Local State. *Log. Methods Comput. Sci.* 4, 4 (2008). https://doi.org/10.2168/LMCS-4(4:2)2008

Received 2025-02-27; accepted 2025-06-27