Stateful Structural Operational Semantics

Sergey Goncharov
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Stefan Milius
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Stelios Tsampas
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Henning Urbat
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

Compositionality of denotational semantics is an important concern in programming semantics. Mathematical operational semantics in the sense of Turi and Plotkin guarantees compositionality, but seen from the point of view of stateful computation it applies only to very fine-grained equivalences that essentially assume unrestricted interference by the environment between any two statements. We introduce the more restrictive *stateful SOS* rule format for stateful languages. We show that compositionality of two more coarse-grained semantics, respectively given by assuming read-only interference or no interference between steps, remains an undecidable property even for stateful SOS. However, further restricting the rule format in a manner inspired by the *cool* GSOS formats of Bloom and van Glabbeek, we obtain the *streamlined* and *cool* stateful SOS formats, which respectively guarantee compositionality of the two more abstract equivalences.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Structural Operational Semantics, Rule Formats, Distributive Laws

Related Version *Extended paper with full proofs*: https://arxiv.org/abs/2202.10866

Funding *Sergey Goncharov*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 215418801
Stefan Milius: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 259234802
Lutz Schröder: Work supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 (grant number 393541319/GRK2475/1-2019)
Stelios Tsampas: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 419850228
Henning Urbat: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 419850228

1 Introduction

A key prerequisite for modular reasoning about process calculi and programming languages is *compositionality*: A denotational semantics is compositional if the associated semantic equivalence forms a congruence, that is, subterms of a given process or program term may be replaced with equivalent subterms without affecting the overall denotational meaning of the term. For instance, the classical GSOS format of Bloom et al. [9] provides a unified formal representation of process languages interpreted over non-deterministic labelled transition systems, and guarantees that bisimilarity is compositional. Similarly, syntactic restrictions

of the GSOS format due to Bloom [8] and van Glabbeek [40] guarantee compositionality for coarser equivalences.

More abstractly, GSOS is captured in Turi and Plotkin's bialgebraic framework of *mathematical operational semantics* [39], in which sets of operational semantic rules are represented as distributive laws of a monad over a comonad, a principle that has come to be used in widely varying semantic settings [6, 22, 13, 24]. In particular, Turi and Plotkin demonstrated that GSOS rules correspond precisely to natural transformations of type

$$\varrho_X \colon \Sigma(X \times (\mathscr{P}_\omega X)^L) \to (\mathscr{P}_\omega \Sigma^\star X)^L,$$

where Σ is a polynomial functor on the category of sets (representing the signature of the process language at hand), L is a set of (transition) labels, $\mathscr{P}_\omega$ is the finite power set functor, corresponding to finitary non-determinism, and $\Sigma^\star$ denotes the free (term) monad on Σ. This is an instance of an *abstract GSOS law*, a natural transformation of type $\Sigma(\mathrm{Id} \times T) \Longrightarrow T\Sigma^\star$, with T, the *behaviour functor*, instantiated to the functor $\mathscr{P}_\omega^L$, which is associated with image-finite L-labelled transition systems.

There is long-standing interest in SOS style specifications of stateful programming languages [33]. The natural instantiation of mathematical operational semantics to this setting would use $TX = (S \times (X + 1))^S$ as the behaviour functor (for a given set S of *states*). This gives rise to an extremely expressive rule format: In abstract GSOS laws of type $\Sigma(\mathrm{Id} \times T) \Longrightarrow T\Sigma^\star$, program constructs receive their arguments as full-blown state transformers, which in particular they can execute or probe on any number of input states. The semantic domain provided by mathematical operational semantics in this case is the final coalgebra for T, which consists of possibly infinite S-branching, S-labelled trees, and thus is an instance of *(coalgebraic) resumption semantics* [30], originally developed for concurrent settings [14, 11]. The induced notion of semantic equivalence, for which the format guarantees compositionality, is very fine-grained: Being a resumption semantics, it assumes that programs cede complete control to the environment between any two consecutive steps, and thus makes rather few programs equivalent. Capturing less sceptical semantics, such as standard sequential end-to-end net execution, in a compositional manner has proved rather more challenging; generally speaking, compositionality is harder for coarser equivalences because less information is available about the behaviour of subterms [40].

In the present work, we approach this problem by restricting the rule format to various degrees. We first note that the operational rules typically associated to imperative languages resemble GSOS rules with an additional input parameter, the present state. We correspondingly introduce the *stateful SOS* format for the specification of stateful languages, and show that stateful SOS specifications are in an one-to-one correspondence with natural transformations of type

$$\delta_X \colon S \times \Sigma(X \times S \times (X + 1)) \to S \times (\Sigma^\star X + 1).$$

In a small-step operational semantics given in terms of transitions on pairs consisting of states in S and program terms (or a termination marker $\checkmark \in 1$), δ_X assigns to a given state (in S) and a program construct applied to argument variables with given next-step operational behaviour (i.e. an element of $\Sigma(X \times S \times (X + 1))$) its small-step operational behaviour. Effectively, this means that, in small-step operational semantics, program constructs can execute and probe their arguments only on the current state. We give a resumption semantics (over the final coalgebra for T as above) for stateful SOS, and show that this semantics agrees with the one obtained by converting δ into a GSOS law, in particular is compositional.

We go on to define two successive coarsenings of resumption semantics: *Trace semantics* assumes that the environment can observe but not manipulate states reached in between

	$(\mathtt{x} := 1;\ \mathtt{y} := \mathtt{x}) = (\mathtt{x} := 1;\ \mathtt{y} := 1)$	$(\mathtt{x} := 1;\ \mathtt{x} := 2) = (\mathtt{x} := 2)$
$\nu\gamma.\,(S \times (\gamma + 1))^S$	✘	✘
$(S^+ + S^\omega)^S$	✔	✘
$(S + 1)^S$	✔	✔

Table 1 Separating denotational domains by program equivalences.

successive computation steps, and correspondingly uses the semantic domain $(S^+ + S^\omega)^S$, the set of functions expecting an initial state and returning a possibly terminating S-stream. The, yet coarser, *termination semantics* additionally abstracts from the intermediate states of a computation, and thus is defined over the semantic domain $(S+1)^S$, the set of functions expecting an initial state and returning either a final state or divergence. Trace semantics has been used, e.g., in the type-theoretic semantics of program logics [26] and in formalizing concurrent systems that feature memory isolation mechanisms [28, 29]. Termination semantics is the semantic domain typically associated with *big-step* [23, 27] or *natural* semantics [19], and is a popular choice in settings where fine architectural details are less relevant [34, 32, 31]. Table 1 presents the three domains in decreasing order of granularity and illustrates their differences in terms of the programs they distinguish. Here, S is the set of variable stores assigning to every program variable its current value. First, consider the programs $\mathtt{x}{:=}1;\ \mathtt{x}{:=}2$ and $\mathtt{x} := 2$. These are clearly equivalent in termination semantics but not in trace semantics, as the additional initial step of the first program is visible in trace semantics. Similarly, the programs $\mathtt{x} := 1;\ \mathtt{y} := \mathtt{x}$ and $\mathtt{x} := 1;\ \mathtt{y} := 1$ are clearly equivalent under trace semantics but not under resumption semantics, as the latter assumes that the value of $\mathtt{x}$ may be changed by the environment between the two steps. In fact, we show as our first main result that despite the restricted expressiveness, it is undecidable whether the coarser program equivalences are compositional for a given stateful SOS specification. In a subsequent step, we thus introduce two sets of syntactic restrictions in the spirit of Bloom [8] and van Glabbeek [40], and show that these guarantee that stateful SOS specifications have compositional trace semantics or termination semantics, respectively.

Related Work The above-mentioned *cool* GSOS rules of Bloom [8] and van Glabbeek [40] guarantee compositionality w.r.t. various flavours of weak bisimilarity; they motivate the *cool stateful SOS* format we introduce here. In a similar vein, Tsampas et al. [37] present abstract compositionality criteria for weak bisimilarity in the context of mathematical operational semantics [38]. Weak bisimilarity is still rather finer than the main semantics of interest for the present work (trace semantics and termination semantics), as it only abstracts away from steps that do not modify the state, such as `skip`.

Abou-Saleh and Pattinson [1, 2] consider abstract GSOS specifications for while-languages and construct semantics in Kleisli categories, working at a somewhat higher level of generality than we do here, in particular parametrizing over notions of side-effect. Roughly speaking, the coarsest of their semantics amounts to a *steps-until-termination* semantics that counts but does not enumerate intermediate states, and thus is coarser than trace semantics but finer than termination semantics. They propose an abstract *condition on cones* [1, Sec. 4.4] that guarantees compositionality for steps-until-termination semantics. This condition is hard to verify in concrete instances but ensured by *evaluation-in-context* rule formats [2] that correspond roughly to our *cool* stateful SOS format, for which we show compositionality even w.r.t. termination semantics (a goal explicitly mentioned by Abou-Saleh and Pattinson [2,

Section 6]). Our *streamlined* stateful SOS format, which guarantees compositionality of trace semantics, appears to be more permissive than evaluation-in-context.

Bloom and Vandraager [10] and Mousavi et al. [25] propose further SOS-style formats for computations with data and prove compositionality results for semantic equivalences resembling our resumption semantics. We note that these results require fairly tedious proofs; this again highlights the advantage of the categorical approach where they come entirely for free (see Theorem 4.6). The *Sfisl* format [25] is shown to make trace semantics compositional, but in contrast to our streamlined format it is not expressive enough to cover a fully fledged while-language. Termination semantics is not considered in either of these works.

2 Preliminaries

We assume that readers are familiar with basic notions from category theory such as functors, natural transformations, and monads. In the following we briefly recall some terminology concerning algebras and coalgebras. Throughout, **Set** denotes the category of sets and functions. We write $1 = \{*\}$ for the terminal object. For a pair X_1, X_2 of objects we write $X_1 \times X_2$ for the product with the projections $\mathsf{fst}\colon X_1 \times X_2 \to X_1$ and $\mathsf{snd}\colon X_1 \times X_2 \to X_2$. For a pair of morphisms $f_i\colon Y \to X_i$, $i = 1, 2$, we let $\langle f_1, f_2\rangle\colon Y \to X_1 \times X_2$ denote the unique induced morphism. The *canonical strength* of an endofunctor $F\colon \mathbf{Set} \to \mathbf{Set}$ is the natural transformation with components $\mathsf{st}_{X,Y}\colon X \times FY \to F(X \times Y)$ defined by $\mathsf{st}_{X,Y}(x, p) = F(\lambda y.\,(x, y))(p)$. We usually drop the subscripts X and Y.

Algebras Given an endofunctor F on a category $\mathcal{C}$, an *F-algebra* is a pair (A, α) of an object A (the *carrier* of the algebra) and a morphism $\alpha\colon FA \to A$ (its *structure*). A *homomorphism* from an F-algebra (A, α) to an F-algebra (B, β) is a morphism $h\colon A \to B$ of $\mathcal{C}$ such that $h \cdot \alpha = \beta \cdot Fh$. Algebras for F and their homomorphims form a category $\mathsf{Alg}\, F$, and an *initial* F-algebra is simply an initial object in that category. If it exists, we denote the initial F-algebra by μF and its structure by $\iota\colon F(\mu F) \to \mu F$.

A common example of functor algebras are algebras over a signature. An *algebraic signature* consists of a set Σ of operation symbols together with a map $\mathsf{ar}\colon \Sigma \to \mathbb{N}$ associating to every operation symbol f its *arity* $\mathsf{ar}(\mathsf{f})$. Symbols of arity 0 are called *constants*. Every signature Σ induces the polynomial functor $\coprod_{\mathsf{f}\in\Sigma}(-)^{\mathsf{ar}(\mathsf{f})}$ on **Set**, which we denote by the same letter Σ. An algebra for the functor Σ then is precisely an algebra for the signature Σ, i.e. a set A equipped with an operation $\mathsf{f}_A\colon A^n \to A$ for every n-ary operation symbol $\mathsf{f} \in \Sigma$. Homomorphisms between Σ-algebras are maps respecting the algebraic structure.

Given a set X of variables, we write $\Sigma^\star X$ for the Σ-algebra of terms generated by Σ with variables from X. It is the *free Σ-algebra* on X, that is, every map $f\colon X \to A$ into the carrier of a Σ-algebra (A, α) uniquely extends to a homomorphism $\bar{f}\colon \Sigma^\star X \to A$. In particular, the free algebra on the empty set is the initial algebra $\mu\Sigma$; it is formed by all *closed terms* of the signature. As shown by Barr [5], the formation of free algebras extends to a monad $\Sigma^\star\colon \mathbf{Set} \to \mathbf{Set}$, the *free monad* on Σ. For every Σ-algebra (A, α) we obtain an Eilenberg-Moore algebra $\widehat{\alpha}\colon \Sigma^\star A \to A$ as the free extension of id_A. This is the map evaluating terms over A in the algebra.

Coalgebras A *coalgebra* for an endofunctor F on $\mathcal{C}$ is a pair (C, γ) of an object C (the *carrier*) and a morphism $\gamma\colon C \to FC$ (its *structure*). A *homomorphism* from an F-coalgebra (C, γ) to an F-coalgebra (D, δ) is a morphism $h\colon C \to D$ such that $Fh \cdot \gamma = \delta \cdot h$. Coalgebras for F and their homomorphisms form a category $\mathsf{Coalg}\, F$, and a *final* coalgebra is a final

object in that category. If it exists, we denote the final F-coalgebra by νF and its structure by $\tau\colon \nu F \to F(\nu F)$, and we write $\gamma^\sharp\colon (C,\gamma) \to (\nu F, \tau)$ for the unique homomorphism.

▶ **Example 2.1.** 1. Fix a set S. The set functor $BX = S \times (X+1)$ has a final coalgebra carried by $\nu B = S^+ + S^\omega$, the set of all non-empty possibly terminating S-streams. Its coalgebra structure $S^+ + S^\omega \to S \times (S^+ + S^\omega + 1)$ sends a stream sw (where $s \in S$ and $w \in S^* + S^\omega$) to (s,w) if $w \in S^+ + S^\omega$ and to $(s,*)$ if w is empty.

2. Similarly, for the set functor $TX = (BX)^S = (S \times (X+1))^S$, the terminal coalgebra is carried by the set of possibly infinite S-ary trees (i.e. every node is either a leaf or has an S-indexed set of children) that have more than one node and where every edge is labelled by an element of S. The coalgebra structure $\nu T \to (S \times (\nu T + 1))^S$ sends a tree t to the map $s \mapsto (s', t')$ where s' is the label of the edge from the root to its s-th child, and t' is the subtree rooted at that child if it has more than one node, or $*$ otherwise.

3 Stateful SOS Specifications

We start off with an observation on the standard operational semantics for sequential composition in imperative languages (see e.g. Plotkin [33]), given by the following rules:

$$\texttt{seq1}\frac{s, p \downarrow s'}{s, (p\,;\, q) \to s', q} \qquad \texttt{seq2}\frac{s, p \to s', p'}{s, (p\,;\, q) \to s', (p'\,;\, q)} \tag{3.1}$$

Rule `seq1` asserts that if a program p, on input (state) s, terminates and produces a new state s', then the program $p\,;\,q$, on input state s, evolves to program q and produces the new state s'. The other case is captured by rule `seq2`, which asserts that if p, on input s, transitions to p' and produces s', then $p\,;\,q$, on input s, transitions to $p'\,;\,q$ and produces s'. Note that for both rules, the *input* s is the same in the premiss and in the conclusion. Consequently, to decide how $p\,;\,q$ transitions from s in the next step, we need to know only how p behaves on s, which we can regard as the input of the entire rule. This allows us to give a concise categorical formulation of the rules `seq1` and `seq2` in terms of a natural transformation $S \times (X \times S \times (X+1))^2 \to (S \times \Sigma^\star X + 1)$ where Σ is a signature containing the binary operation symbol '$;$'. The transformation is defined by

$$(s, (x, s', *), (y, \_, \_)) \mapsto (s', y) \qquad \text{and} \qquad (s, (x, s', x'), (y, \_, \_)) \mapsto (s', (x'\,;\, y)).$$

Compare the above with the interpretation obtained by instantiating the GSOS principle [39] to stateful computations in the standard manner [38]. The interpretation of '$;$' is then given as a natural transformation $(X \times (S \times (X+1))^S)^2 \to (S \times (\Sigma^\star X + 1))^S$ whose uncurried form $S \times (X \times (S \times (X+1))^S)^2 \to S \times (\Sigma^\star X + 1)$ is defined by

$$(s, (x, f), (y, \_)) \mapsto \begin{cases} (s', y) & \text{if } f(s) = (s', *), \\ (s', (x'\,;\, y)) & \text{if } f(s) = (s', x'). \end{cases}$$

In this setting, the semantics of $p\,;\,q$ receives the next-step behaviours of p, q as state transformers, and can in principle probe these state transformers on arbitrary states (of course, for '$;$', this does not actually happen). By contrast, our rule format, the stateful SOS format formally introduced next, embodies the restriction that the behaviour of a complex term on an input state s is predicated only on the behaviour of its subterms on s. It is this trade-off in expressiveness that buys our compositionality results for stateful SOS specifications.

The Stateful SOS Rule Format We proceed to underpin the intuition given above with formal definitions. We fix a countably infinite set $\mathcal{V} = \{x_1, x_2, \dots\} \cup \{y_1, y_2, \dots\}$ of (*meta-*)*variables* and a countable set S of *states*; in typical applications the elements of S are variable stores. Moreover, we fix an algebraic signature Σ, equivalently a polynomial functor also denoted Σ (cf. Section 2). We think of the operations in Σ as program constructs, and correspondingly, *programs* are closed Σ-terms, i.e. terms formed using only the operations in Σ, with constants in Σ forming the base case.

▶ **Definition 3.1** (Literals)**.** A *progressing* Σ*-literal* is an expression $s, p \to s', q$ with $p, q \in \Sigma^\star\mathcal{V}$ and $s, s' \in S$. We say that s is the *input*, p is the *source*, s' is the *output* and q is the *target* of the literal. A *terminating* Σ*-literal* is an expression $s, p \downarrow s'$ with $s, s' \in S$ and $p \in \Sigma^\star\mathcal{V}$. In this case, s is the input, p is the source and s' is the output of the literal. A Σ*-literal* (without further qualification) is either a progressing or a terminating Σ-literal.

Our rule format shares some similarities with *stream GSOS* [20, Def. 37].

▶ **Definition 3.2** (Rules)**.** A *stateful SOS rule* for an n-ary operator $\mathsf{f} \in \Sigma$ is an expression

$$\frac{l_1 \quad \dots \quad l_n}{L} \tag{3.2}$$

(or, in inline notation, $l_1 \ \dots \ l_n / L$) where $l_1, \dots, l_n$ (the *premisses* of the rule) and L (the *conclusion* of the rule) are Σ-literals that have the same input $s \in S$, the *input* of the rule, and satisfy the following conditions:

1. The source of the premiss l_j is the variable x_j, and the target is y_j if l_j is progressing.
2. The source of the conclusion L is the term $\mathsf{f}(x_1, \dots, x_n)$. Moreover, if L is progressing, the variables of its target term appear either as the source or the target of some premiss.

The rule is *progressing* if L is progressing, and otherwise the rule is *terminating*. The *trigger* of the rule is the tuple formed by its input s together with the sequence of pairs $\overrightarrow{(s', \mathsf{c})} = (s'_1, \mathsf{c}_1), \dots, (s'_n, \mathsf{c}_n)$, where s'_j is the output of l_j and $\mathsf{c}_j \in \{\mathtt{pr}, \mathtt{te}\}$ indicates whether l_j is progressing ($c_j = \mathtt{pr}$) or terminating ($c_j = \mathtt{te}$).

▶ **Definition 3.3.** A *stateful SOS specification* is a set of stateful SOS rules such that for each n-ary operator f, each $s \in S$ and each sequence $\overrightarrow{(s', \mathsf{c})} = (s'_1, \mathsf{c}_1), \dots, (s'_n, \mathsf{c}_n)$ where $s'_j \in S$ and $\mathsf{c}_j \in \{\mathtt{pr}, \mathtt{te}\}$, there is exactly one rule for f with trigger $(s, \overrightarrow{(s', \mathsf{c})})$.

▶ **Notation 3.4.** By writing

$$\frac{l_1 \quad \dots \quad l_{j-1} \quad l_{j+1} \quad \dots \quad l_n}{L}$$

we mean the set of all stateful SOS rules of the form $l_1 \ \dots \ l_n / L$ (with the missing premiss l_j filled in in any way possible). This captures the situation where the behaviour of the source $\mathsf{f}(x_1, \dots, x_n)$ of L does not depend on the behaviour of x_j, given $l_1, \dots, l_{j-1}, l_{j+1}, \dots, l_n$.

▶ **Remark 3.5.** The use of fixed enumerated variables $x_1, x_2, \dots$ and $y_1, y_2, \dots$ simplifies abstract reasoning about stateful SOS (e.g. Theorem 3.9 below). In examples, we use arbitrary variable names such as p, q, x, y, and we typically write rules using rule schemes, using hopefully self-explanatory notation. For instance, rule `seq1` in Figure 1 (discussed in detail in Example 3.6) is to be understood as the set $\{\ s, p \downarrow s' \ /\ s, (p\,;\,q) \to s', q \mid s, s' \in S\}$ of stateful SOS rules, with variables p, q, and rule `while1` as the set $\{\ /s, \mathtt{while}\ e\ p \downarrow s \mid s \in S, [e]_s = 0\}$ (with premiss omitted as per Notation 3.4). Note the side condition $[e]_s = 0$ (expression e evaluates to 0 in state s) of `while1`; the rule schemes and their side conditions need to be

$$\texttt{skip}\ \frac{}{s, \texttt{skip} \downarrow s} \qquad \texttt{asn}\ \frac{}{s, (x := e) \downarrow s_{[x \leftarrow [e]_s]}}$$

$$\texttt{while1}\ \frac{}{s, \texttt{while}\ e\ p \downarrow s}\ [e]_s = 0 \qquad \texttt{while2}\ \frac{}{s, \texttt{while}\ e\ p \to s, (p;\ \texttt{while}\ e\ p)}\ [e]_s \neq 0$$

$$\texttt{seq1}\ \frac{s, p \downarrow s'}{s, (p;\ q) \to s', q} \qquad \texttt{seq2}\ \frac{s, p \to s', p'}{s, (p;\ q) \to s', (p';\ q)}$$

Figure 1 Operational semantics of *While*.

set up in such a way that they actually obey the restrictions in Definition 3.3. For example, in the case of `while1` and `while2`, this is ensured by the respective side conditions ($[e]_s = 0$ and $[e]_s \neq 0$) being exhaustive and mutually exclusive.

▶ **Example 3.6.** We will use a prototypical imperative language, *While*, as a running example. Fix a countably infinite set $\mathcal{A}$ of program variables; then, the set S of *stores* consists of all maps $s\colon \mathcal{A} \to \mathbb{N}$ whose *support* $\{x \in \mathcal{A} \mid s(x) \neq 0\}$ is finite. We denote by $s_{[x \leftarrow v]}$ the result of changing the value of variable x to v in a store s. Moreover, we assume a set E of expressions that include the arithmetic operations $+, -, *$, constants $n \in \mathbb{N}$ and variables $x \in \mathcal{A}$. We write $[e]_s$ for the evaluation of expression e under store s (in the literature, evaluation is often defined stepwise by induction on the structure of the expression [33]; since this process does not affect the program state, we instead assume a denotational semantics for simplicity). The syntax of *While* is given by the grammar

$$\langle \text{prog} \rangle \quad ::= \quad \texttt{skip} \mid x := e \mid \langle \text{prog} \rangle ; \langle \text{prog} \rangle \mid \texttt{while}\ e\ \langle \text{prog} \rangle \qquad (x \in \mathcal{A},\ e \in E),$$

which in terms of algebraic operations means that the signature Σ includes constants `skip` and $x := e$ for all $x \in \mathcal{A}$, $e \in E$, a binary operation `;` and a unary operation `while` e for each $e \in E$. The corresponding polynomial functor is

$$\Sigma X = 1 + \mathcal{A} \times E + X \times X + E \times X.$$

The operational semantics of *While* in the form of a stateful SOS specification is shown in Figure 1, using rule schemes as per Remark 3.5.

As indicated by the discussion at the beginning of this section, stateful SOS specifications can be represented as natural transformations:

▶ **Definition 3.7.** A *stateful SOS law* is a natural transformation

$$\delta_X\colon S \times \Sigma(X \times S \times (X+1)) \to S \times (\Sigma^\star X + 1) \qquad (X \in \mathbf{Set}).$$

▶ **Remark 3.8.** 1. Every stateful SOS specification $\mathcal{L}$ yields a stateful SOS law

$$\delta_X = [\delta_X^{\mathsf{f}}]_{\mathsf{f} \in \Sigma}\colon S \times \Sigma(X \times S \times (X+1)) \to S \times (\Sigma^\star X + 1) \qquad (X \in \mathbf{Set})$$

by distributing $S \times (-)$ over $\Sigma(X \times S \times (X+1))$ and copairing the maps

$$\delta_X^{\mathsf{f}}\colon S \times (X \times S \times (X+1))^{\mathsf{ar}(\mathsf{f})} \to S \times (\Sigma^\star X + 1) \qquad (\mathsf{f} \in \Sigma) \tag{3.3}$$

defined as follows. Given $(s, ((v_1, s_1', w_1), \dots, (v_n, s_n', w_n))) \in S \times (X \times S \times (X+1))^n$ with $n = \mathsf{ar}(f)$, let $l_1\ \dots\ l_n/L$ be the unique rule in $\mathcal{L}$ with source f and trigger

$(s, ((s'_1, c_1), \dots, (s'_n, c_n))$ where $c_j = \mathtt{pr}$ if $w_j \in X$ and $c_j = \mathtt{te}$ if $w_j = *$. Let s' be the output of L. Then $\delta^{\mathsf{f}}_X(s, ((v_1, s'_1, w_1), \dots, (v_n, s'_n, w_n)))$ is $(s', *)$ if the rule is terminating, and otherwise (s', t') where $t' \in \Sigma^\star X$ is the term obtained from the target $t \in \Sigma^\star \mathcal{V}$ of L by substituting x_j by v_j and y_j by w_j (the latter whenever $c_j = \mathtt{pr}$).

2. Conversely, every stateful SOS law δ yields a stateful SOS specification $\mathcal{L}$ whose rules are defined as follows. For every n-ary operation symbol $\mathsf{f} \in \Sigma$, $s, s'_1, \dots, s'_n \in S$ and $W \subseteq \{1, \dots, n\}$, let (s', t) be the value of $\delta^{\mathsf{f}}_{\mathcal{V}}$ on $(s, ((x_1, s'_1, w_1), \dots, (x_n, s'_n, w_n)))$ where $w_j = y_j$ if $j \in W$ and $w_j = *$ otherwise. If $t \in \Sigma^\star \mathcal{V}$, then $\mathcal{L}$ contains the rule

$$\frac{(s, x_j \to s'_j, y_j)_{j \in W} \qquad (s, x_j \downarrow s'_j)_{j \in \{1,\dots,n\} \setminus W}}{s, \mathsf{f}(x_1, \dots, x_n) \to s', t},$$

and if $t = *$, then $\mathcal{L}$ contains the rule

$$\frac{(s, x_j \to s'_j, y_j)_{j \in W} \qquad (s, x_j \downarrow s'_j)_{j \in \{1,\dots,n\} \setminus W}}{s, \mathsf{f}(x_1, \dots, x_n) \downarrow s'}.$$

▶ **Theorem 3.9.** *There is a bijective correspondence between (1) stateful SOS specifications, (2) stateful SOS laws, and (3) families of maps of the form*

$$\big(r_{\mathsf{f},W} \colon S \times S^{\mathsf{ar}(\mathsf{f})} \to S \times \Sigma^\star(\mathsf{ar}(\mathsf{f}) + W) + S\big)_{\mathsf{f} \in \Sigma, W \subseteq \mathsf{ar}(\mathsf{f})}.$$

Here we identify the natural number $\mathsf{ar}(\mathsf{f})$ *with the set* $\{1, \dots, \mathsf{ar}(\mathsf{f})\}$.

The correspondence between (1) and (2) is given by the translations of Remark 3.8, and the correspondence between (2) and (3) is shown using the Yoneda lemma.

4 Categorical Semantics and Compositionality

We proceed to develop a categorical treatment of stateful SOS along the lines of mathematical operational semantics in the style of Turi and Plotkin [39] and Bartels [6]. Furthermore, we shall define two semantic domains of interest, both coarser than the one initially obtained through Turi-Plotkin semantics, and show that the problem of whether a given stateful SOS specification is compositional is undecidable. We recall that if the denotational semantics of a programming language is given by a map $[\![-]\!] \colon \mu\Sigma \to D$ into a semantic domain D, then it is called *compositional* if the corresponding behavioural equivalence forms a congruence, that is, for every n-ary operator $\mathsf{f} \in \Sigma$ and programs $p_i, q_i \in \mu\Sigma$ $(i = 1, \dots, n)$,

$$[\![p_i]\!] = [\![q_i]\!] \text{ for } i = 1, \dots, n \qquad \text{implies} \qquad [\![\mathsf{f}(p_1, \dots, p_n)]\!] = [\![\mathsf{f}(q_1, \dots, q_n)]\!].$$

Compositionality asserts that subprograms of a program p may be replaced with equivalent subprograms without affecting the semantics of p, and thus allows modular reasoning.

4.1 GSOS Laws

Turi and Plotkin's *mathematical operational semantics* [39] identifies sets of rules in structural operational semantics (SOS) with distributive laws of various types on a cartesian base category. We will work more specifically with distributive laws of free monads over cofree copointed functors on the base category **Set**, where the free monad is associated to a polynomial functor. Such distributive laws can equivalently be presented as follows.

▶ **Definition 4.1.** Given a polynomial functor Σ and an endofunctor T on **Set**, a *GSOS law of* Σ *over* T is a natural transformation $\varrho \colon \Sigma(\mathrm{Id} \times T) \Longrightarrow T\Sigma^\star$.

We shall see below that stateful SOS laws determine GSOS laws. The interested reader may find further examples of GSOS laws in the literature [38, 6, 20]. Roughly speaking, the input of ϱ is a (program) operation applied to pairs each consisting of a meta-variable and its assumed next-step behaviour (encapsulated in T), and the output is a next-step behaviour reaching poststates given as programs with meta-variables.

Given a GSOS law ϱ, the initial Σ-algebra can be equipped with a unique T-coalgebra structure $\gamma\colon \mu\Sigma \to T(\mu\Sigma)$ such that the diagram

$$\begin{array}{ccccc} \Sigma(\mu\Sigma) & & \xrightarrow{\quad\iota\quad} & & \mu\Sigma \\ {\scriptstyle \Sigma\langle \mathsf{id},\gamma\rangle}\downarrow & & & & \downarrow{\scriptstyle \gamma} \\ \Sigma(\mu\Sigma \times T(\mu\Sigma)) & \xrightarrow{\varrho_{\mu\Sigma}} & T\Sigma^\star(\mu\Sigma) & \xrightarrow{T\hat{\iota}} & T(\mu\Sigma) \end{array} \tag{4.1}$$

commutes (see Section 2 for the notation). The coalgebra $(\mu\Sigma, \gamma)$ is called the *operational model* of ϱ. Dually, assuming the existence of a final coalgebra νT, there is a unique Σ-algebra structure $\alpha\colon \Sigma(\nu T) \to \nu T$ such that the following diagram commutes:

$$\begin{array}{ccccc} \Sigma(\nu T) & \xrightarrow{\Sigma\langle \mathsf{id},\tau\rangle} & \Sigma(\nu T \times T(\nu T)) & \xrightarrow{\varrho_{\nu T}} & T\Sigma^\star(\nu T) \\ {\scriptstyle \alpha}\downarrow & & & & \downarrow{\scriptstyle T\hat{\alpha}} \\ \nu T & & \xrightarrow{\quad\tau\quad} & & T(\nu T) \end{array} \tag{4.2}$$

The algebra $(\nu T, \alpha)$ is the *denotational model* of ϱ. A fundamental well-behavedness property of GSOS laws is that the unique Σ-algebra homomorphism $(\mu\Sigma, \iota) \to (\nu T, \alpha)$ and the unique T-coalgebra homomorphism $(\mu\Sigma, \gamma) \to (\nu T, \tau)$ coincide. We denote this morphism by

$$\mathsf{beh}_\varrho\colon \mu\Sigma \to \nu T, \tag{4.3}$$

and we think of it as assigning to programs their denotational behaviour. Compositionality of this semantics is immediate from the fact that beh_ϱ is a Σ-algebra homomorphism.

4.2 Semantic Domains for Stateful SOS

We proceed to introduce three denotational semantics of stateful SOS, in order of increasing abstraction: *resumption semantics*, in which the program essentially cedes control to the environment between any two program steps; *trace semantics*, where the environment may observe but not manipulate the state between program steps; and *termination semantics*, in which only the effect of executing the program end-to-end is observable.

▶ **Notation 4.2.** From now on, we instantiate the functor T of Definition 4.1 to

$$TX = (S \times (X + 1))^S,$$

for a fixed set S of states. Thus T represents state transformers with possible non-termination.

Resumption semantics Every stateful SOS law δ (see Definition 3.7) canonically induces a GSOS law

$$\hat{\delta}\colon \Sigma(\mathrm{Id} \times T) \Longrightarrow T\Sigma^\star.$$

This will guarantee compositionality for the most fine-grained of our semantics, which we shall refer to as *resumption semantics*, via established methods of mathematical operational

semantics as recalled above. Details are as follows. The component $\hat{\delta}_X$ is obtained by currying the composite

$$\begin{aligned} S \times \Sigma(X \times TX) &\xrightarrow{\langle \mathsf{fst}, \mathsf{st} \rangle} S \times \Sigma(S \times (X \times TX)) \cong S \times \Sigma(X \times (S \times TX)) \\ &\xrightarrow{\mathsf{id} \times \Sigma(\mathsf{id} \times \mathsf{ev})} S \times \Sigma(X \times S \times (X+1)) \xrightarrow{\delta_X} S \times (\Sigma^\star X + 1), \end{aligned} \tag{4.4}$$

where $\mathsf{st}\colon S \times \Sigma(X \times TX) \to \Sigma(S \times (X \times TX))$ is the strength (cf. Section 2) and $\mathsf{ev}\colon S \times TX = S \times (S \times (X+1))^S \to S \times (X+1)$ denotes the evaluation map. Recall from Example 2.1 that the final coalgebra for T is carried by the set of possibly infinite S-branching trees, with edges labelled in S. Using (4.1) we obtain the operational model $\gamma\colon \mu\Sigma \to T(\mu\Sigma)$ associated to $\hat{\delta}$. In terms of stateful SOS specifications, it can be described as follows.

▶ **Definition 4.3.** Given a stateful SOS specification $\mathcal{L}$, its *transition function* is the map

$$\gamma_0\colon S \times \mu\Sigma \to S \times (\mu\Sigma + 1)$$

inductively defined by

$$\gamma_0(s, \mathsf{f}(t_1, \ldots, t_n)) = m(\delta^{\mathsf{f}}_{\mu\Sigma}(s, (d_1, \ldots, d_n)))$$

where

$$d_j = (t_j, \gamma_0(s, t_j)) \qquad \text{and} \qquad m = \big(S \times (\Sigma^\star(\mu\Sigma) + 1) \xrightarrow{\mathsf{id} \times (\hat{\iota} + \mathsf{id})} S \times (\mu\Sigma + 1)\big),$$

using the term evaluation map $\hat{\iota}\colon \Sigma^\star(\mu\Sigma) \to \mu\Sigma$, and $\delta^{\mathsf{f}}_{\mu\Sigma}$ as in (3.3). Thus, $\gamma_0(s, p)$ performs the first computation step of program p on input s according to the specification $\mathcal{L}$. We write

$$s, p \to s', p' \qquad \text{and} \qquad s, p \downarrow s'$$

if $\gamma_0(s, p) = (s', p')$ and $\gamma_0(s, p) = (s', *)$, respectively.

▶ **Proposition 4.4.** *Let $\mathcal{L}$ be a stateful SOS specification with its associated transition function γ_0 and operational model γ. Then*

$$\gamma = \mathsf{curry}(\gamma_0)\colon \mu\Sigma \to (S \times (\mu\Sigma + 1))^S.$$

The proof makes use of an induction principle that combines *primitive recursion* (see e.g. [17, Prop. 2.4.7]) and *induction with parameters* (see e.g. [17, Exercise 2.5.5]).

▶ **Definition 4.5.** The *resumption semantics* of a stateful SOS specification $\mathcal{L}$ is given by

$$[-]_{\mathcal{L}} = \mathsf{beh}_{\hat{\delta}}\colon \mu\Sigma \to \nu T,$$

where δ is the stateful SOS law associated to $\mathcal{L}$, $\hat{\delta}$ is as per (4.4), and beh is defined in (4.3). Let $\sim_{\mathcal{L}}$ denote the corresponding behavioural equivalence, that is, $p \sim_{\mathcal{L}} q$ iff $[p]_{\mathcal{L}} = [q]_{\mathcal{L}}$ for a given pair $p, q \in \mu\Sigma$. We drop subscripts if $\mathcal{L}$ is clear from the context.

Note that since T preserves weak pullbacks, $\sim_{\mathcal{L}}$ coincides with T-bisimilarity in the operational model $\gamma\colon \mu\Sigma \to T(\mu\Sigma)$ [35]. From the discussion in Section 4.1 we immediately get

▶ **Theorem 4.6.** *The resumption semantics of stateful SOS specifications is compositional.*

Resumption semantics is very fine-grained, essentially because it does not pass the output state of a computation step on as the input state of the next step; that is, resumption semantics assumes that the environment takes complete control in between steps. For instance, consider the *While* programs

$$t_1 = \big(\mathtt{x} := 1;\ \mathtt{x} := \mathtt{x} + 1\big) \qquad \text{and} \qquad t_2 = \big(\mathtt{x} := 1;\ \mathtt{x} := \mathtt{x} * 2\big).$$

The resumption semantics of these programs in each case consists in an S-branching tree of depth 2, in which the edge from the root to its s-th child is labelled $s[x \leftarrow 1]$ and the edges at the next level are correspondingly labelled according to the effect of the assignments $\mathtt{x} := \mathtt{x} + 1$ and $\mathtt{x} := \mathtt{x} * 2$, respectively. In particular, the semantics of the two programs differ – as intuitively expected under a resumption semantics, since the environment may manipulate the value of x in between the two assignments. To obtain a more coarse-grained notion of process equivalence, we have to quotient the semantic domain νT further.

Trace Semantics Consider the set functor B given by

$$BX = S \times (X + 1);$$

thus $TX = (BX)^S$. Recall from Example 2.1 that the final coalgebra νB is carried by the set $S^+ + S^\omega$ of possibly terminating S-streams. The set $(\nu B)^S$ serves as the semantic domain for *trace semantics* for imperative programs [26, 28, 29], which associates to a program the possibly terminating sequence of states it computes from a given initial state. In order to formally introduce trace semantics in our setting, we proceed to construct a quotient map $\nu T \twoheadrightarrow (\nu B)^S$ by coinduction. To this end, we define the functor $(\overline{-})\colon \mathsf{Coalg}\, T \to \mathsf{Coalg}\, B$, which maps a T-coalgebra (C, ζ) to the B-coalgebra

$$\bar\zeta = S \times C \xrightarrow{\mathsf{id}\times\zeta} S \times (BC)^S \xrightarrow{\mathsf{ev}} BC = S \times (C+1) \xrightarrow{\langle \mathsf{fst},\mathsf{st}\rangle} S \times (S \times C + 1) = B(S \times C),$$

where $\mathsf{st}\colon S \times (C+1) \to S \times C{+}1$ is the strength of the functor $(\text{–})+1$, given by $(s, c) \mapsto (s, c)$ and $(s, *) \mapsto *$. Intuitively, while $\zeta^\sharp\colon C \to \nu T$ (see Section 2 for the notation) maps a coalgebra state of C to its tree of state transformers, $\overline{\zeta}^\sharp(s, x) \in \nu B$ executes all these state transformers without interruption, beginning at s and feeding the output state of each previous step to the next step, and outputs the intermediate states reached in each step. Applying $(\overline{-})$ to the final coalgebra $(\nu T, \tau)$, we obtain a B-coalgebra $(S \times \nu T, \overline{\tau})$, and currying the unique coalgebra homomorphism $\overline{\tau}^\sharp\colon S \times \nu T \to \nu B$ yields the desired quotient map

$$\mathsf{trc} = \mathsf{curry}(\overline{\tau}^\sharp)\colon \nu T \twoheadrightarrow (\nu B)^S. \tag{4.5}$$

▶ **Proposition 4.7.** *The map* trc *is surjective.*

▶ **Definition 4.8.** The *trace semantics* of a stateful SOS specification $\mathcal{L}$ is given by

$$[\![-]\!]_{\mathcal{L}} = (\mu\Sigma \xrightarrow{[-]_{\mathcal{L}}} \nu T \xrightarrow{\mathsf{trc}} (\nu B)^S).$$

Let $\simeq_{\mathcal{L}}$ denote the corresponding behavioural equivalence, that is, $p \simeq_{\mathcal{L}} q$ iff $[\![p]\!]_{\mathcal{L}} = [\![q]\!]_{\mathcal{L}}$, for $p, q \in \mu\Sigma$. We drop subscripts if $\mathcal{L}$ is clear from the context.

▶ **Remark 4.9.** Equivalently, $[\![-]\!]_{\mathcal{L}}$ is the curried form of the unique B-coalgebra homomorphism from $(S \times \mu\Sigma, \bar\gamma)$ to νB (recall that $(\mu\Sigma, \gamma)$ is the operational model of $\mathcal{L}$). Since

$$\bar\gamma = \big(S \times \mu\Sigma \xrightarrow{\gamma_0} S \times (\mu\Sigma + 1) \xrightarrow{\langle \mathsf{fst},\mathsf{st}\rangle} S \times (S \times \mu\Sigma + 1) = B(S \times \mu\Sigma)\big)$$

by definition of $\bar{\gamma}$ and Proposition 4.4, we see that for every $p \in \mu\Sigma$ and $s \in S$, the possibly infinite stream $[\![p]\!]_{\mathcal{L}}(s) = s_1 s_2 s_3 \cdots$ is the sequence of states computed by the program p on input state s, cf. Definition 4.3:

$$s, p \to s_1, p_1 \to s_2, p_2 \to s_3, p_3 \to \cdots .$$

Hence trace equivalence $p \simeq q$ holds iff for each input state s, programs p and q produce the same sequence of states.

The following example demonstrates that trace semantics is generally not compositional:

▶ **Example 4.10.** We extend *While* by adding a unary operator $\lfloor\cdot\rfloor$ with

$$\frac{s, p \to s', p'}{s, \lfloor p \rfloor \to \emptyset, \lfloor p' \rfloor} \qquad \frac{s, p \downarrow s'}{s, \lfloor p \rfloor \downarrow s'}$$

where $\emptyset$ denotes the store with all variables set to 0. For $t_1 = \big(\mathtt{x} := 1\,;\ \mathtt{x} := \mathtt{x} + 1\big)$ and $t_2 = \big(\mathtt{x} := 1\,;\ \mathtt{x} := \mathtt{x} * 2\big)$, we have that $t_1 \simeq t_2$ but $\lfloor t_1 \rfloor \not\simeq \lfloor t_2 \rfloor$ (since in $\lfloor t_1 \rfloor$ and $\lfloor t_2 \rfloor$, the store is erased after the first assignment).

Termination Semantics As the coarsest of our semantic domains, we shall use the set $(S + \{\bot\})^S \cong (S+1)^S$ of state transformers on S with possible non-termination featuring pervasively in the denotational semantics of imperative programming (e.g. [34, 32, 31]). In comparison to $(\nu B)^S$, this domain abstracts from the intermediate steps of the computation. The essence of this abstraction is captured by the map

$$\mathsf{fn} \colon \nu B \to S + 1 \qquad \text{defined by} \qquad \mathsf{fn}(x) = \begin{cases} s & \text{if } x \text{ is finite, with last state } s, \\ \bot & \text{otherwise.} \end{cases}$$

▶ **Definition 4.11.** The *termination semantics* of a stateful SOS specification $\mathcal{L}$ is given by

$$[\![-]\!]_{\mathcal{L}} = (\mu\Sigma \xrightarrow{[\![-]\!]_{\mathcal{L}}} (\nu B)^S \xrightarrow{\mathsf{fn}^S} (S+1)^S).$$

Let $\approx_{\mathcal{L}}$ denote the corresponding behavioural equivalence, that is, $p \approx_{\mathcal{L}} q$ iff $[\![p]\!]_{\mathcal{L}} = [\![q]\!]_{\mathcal{L}}$ for $p, q \in \mu\Sigma$. We drop subscripts if $\mathcal{L}$ is clear from the context.

Thus $p \approx q$ iff for each initial state s, if p eventually terminates with final state s' then q eventually terminates with final state s' and vice-versa. Termination semantics is generally not compositional: the programs t_1 and t_2 of Example 4.10 satisfy $t_1 \approx t_2$ but $\lfloor t_1 \rfloor \not\approx \lfloor t_2 \rfloor$.

The maps introduced in this section are summarized in the following commutative diagram:

$$\begin{array}{ccccc} & & \mu\Sigma & & \\ & \swarrow^{[-]_{\mathcal{L}}} & \downarrow{\scriptstyle [\![-]\!]_{\mathcal{L}}} & \searrow^{[\![-]\!]_{\mathcal{L}}} & \\ \nu T & \xrightarrow{\mathsf{trc}} & (\nu B)^S & \xrightarrow{\mathsf{fn}^S} & (S+1)^S \end{array} \tag{4.6}$$

4.3 Compositionality is Undecidable

We have seen that in contrast to resumption semantics, both trace and termination semantics generally fail to be compositional. As it turns out, reasoning about compositionality in these two cases is a very complex, viz. undecidable, task.

To make the ensuing decision problems precise, we fix suitable encodings of states and terms as finite strings and regard a stateful SOS specification $\mathcal{L}$ as a total function that assigns to a given operation symbol, input state and list of premisses the target of the conclusion and output state of the respective rule. From a computational point of view, a minimum requirement on every reasonable specification $\mathcal{L}$ is that it admits some finite representation. Hence, for simplicity, we assume in the following theorem that specifications are primitive recursive functions. For instance, this is clearly the case for the *While* language.

▶ **Theorem 4.12.** *It is undecidable whether the trace semantics (or termination semantics, respectively) induced by a primitive recursive stateful SOS specification is compositional.*

Proof sketch. The halting problem reduces to the compositionality problem. The idea is to take programs akin to t_1 and t_2 in Example 4.10 and precompose them with the simulation of a given Turing machine. This can be specified in stateful SOS. The failure of compositionality described in Example 4.10 then occurs if, and only if, the simulated machine halts. ◀

In view of the fact that there is no sound and complete decision procedure for compositionality w.r.t. $\simeq$ and $\approx$, we instead move on to identify easily checked *sound* syntactic criteria that, although necessarily incomplete, are sufficiently broad.

5 Cooling the Stateful SOS Format

We now introduce two sets of restrictions on the stateful SOS rule format, called *streamlined stateful SOS* and *cool stateful SOS*, that guarantee trace and termination semantics, respectively, to be compositional. Our approach is inspired by the work of Bloom [8] and van Glabbeek [40] on the *cool* congruence formats for weak bisimilarity for GSOS specifications. The following definition will help describe the restricted formats. We make pervasive use of the abbreviations from Notation 3.4, and we will additionally employ $s, p \to s', *$ as an alternative notation for a terminating literal $s, p \downarrow s'$.

▶ **Definition 5.1.** Let $\mathcal{L}$ be a stateful SOS specification.

1. An n-ary operator f is *passive* if all rules for f are of the form

$$\frac{}{s, \mathsf{f}(x_1, \dots, x_n) \to s', t} \qquad \text{where } t \in \Sigma^\star(\{x_1, \dots, x_n\}) \text{ or } t = *.$$

   In other words, the one-step behaviour of $\mathsf{f}(x_1, \dots, x_n)$ does not depend on the one-step behaviour of any of its subterms. In particular, every constant is passive. An *active* operator is one which is not passive.
2. A progressing rule for an n-ary operator f is *receiving at position* $j \in \{1, \dots, n\}$ if its j-th premiss $s, x_j \to s', y_j$ is progressing and the variable y_j appears in the target of the conclusion. We say that the rule is *receiving* if it is receiving at some position j.

5.1 Streamlined Stateful SOS

As indicated above, the streamlined Stateful SOS format, introduced next, will guarantee compositionality of trace semantics.

▶ **Definition 5.2.** A stateful SOS specification is *streamlined* if for every active operator f of arity n there exists $j \in \{1, \dots, n\}$ (the *receiving position* of f) such that the following holds:

1. All receiving rules for f are of the form

$$\frac{s, x_j \to s', y_j}{s, \mathsf{f}(x_1, \ldots, x_n) \to s', t} \qquad \text{where } t = \mathsf{f}(x_1, \ldots, x_n)[y_j/x_j] \text{ or } t = y_j;$$

   here, $[u/x]$ denotes substitution of the variable x by the term u.
2. All non-receiving rules for f are of the form

$$\frac{l_1 \quad l_2 \quad \cdots \quad l_n}{s, \mathsf{f}(x_1, \ldots, x_n) \to s', t} \qquad \text{where } t \in \Sigma^\star(\{x_1, \ldots, x_n\} \setminus \{x_j\}) \text{ or } t = *.$$

Note that in a stateful SOS specification, receiving rules for an active operator f are receiving *only* in the receiving position of f. What Definition 5.2 boils down to is that an active operator can only progress its subterm at the receiving position j, leaving everything else unchanged and making sure that the output state in the j-th premiss is correctly propagated, and discards the j-th subterm once it terminates.

▶ **Example 5.3.** The *While* language (cf. Figure 1) is streamlined. The only active operator is sequential composition $p\,;\, q$. Its progressing rules are receiving in the left position, and upon termination the left subterm is discarded.

Further examples are discussed after Corollary 5.5.

▶ **Theorem 5.4.** *Trace semantics is compositional for streamlined stateful SOS specifications.*

Proof sketch. For $p, q \in \mu\Sigma$ and $k \in \mathbb{N}$ we put $p \simeq_k q$ if the programs p and q are k-step trace equivalent, that is, for every $s \in S$ the streams $[\![p]\!](s)$ and $[\![q]\!](s)$ have the same prefix of length at most k. By induction on k one proves $\simeq_k$ to be a congruence, using a judicious strengthening of the inductive claim for receiving positions of active operators. This implies that $\simeq$ is a congruence, whence trace semantics is compositional. ◀

From Theorem 5.4 we can deduce a slightly stronger statement. In what follows, the *kernel* of a map $e\colon X \to Y$ is the equivalence relation on X relating x, x' iff $e(x) = e(x')$.

▶ **Corollary 5.5.** *For every streamlined stateful SOS specification, the kernel of the map* $\mathsf{trc}\colon \nu T \nrightarrow (\nu B)^S$ *is a congruence w.r.t. the canonical* Σ*-algebra structure on* νT *as per* (4.2).

We next look at examples of streamlined specifications but also at a few pathological cases where compositionality breaks.

▶ **Example 5.6.** Streamlined specifications allow for complex control flow over programs, including *signal* or *interrupt handling*. For instance, we can extend *While* by a distinguished variable $\mathtt{i}$ serving as an interrupt flag and modify the rules of sequential composition to

$$\frac{s, p \downarrow s'}{s, (p\,;\, q) \to s', q} \qquad \frac{s, p \to s', p'}{s, (p\,;\, q) \to s', (p'\,;\, q)}\, [\mathtt{i}]_s = 0$$

$$\frac{s, p \to s', p'}{s, (p\,;\, q) \to s', q}\, [\mathtt{i}]_s \neq 0 \wedge P(s') \qquad \frac{s, p \to s', p'}{s, (p\,;\, q) \to s', (p'\,;\, q)}\, [\mathtt{i}]_s \neq 0 \wedge \neg P(s')$$

where $P \subseteq S$. If flag $\mathtt{i}$ is enabled and predicate P is true for the output s' of p, then p is terminated prematurely. This type of rules can also be used to implement *listeners* or *observers* in high-level programming languages [18].

▶ **Example 5.7.** **1.** Recall the operator $\lfloor\cdot\rfloor$ from Example 4.10, which breaks compositionality for trace semantics. The operator is active, and its progressing rule is receiving but does not propagate the output state of its premiss, so the stateful SOS specification of *While* with $\lfloor\cdot\rfloor$ fails to be streamlined (as it must, by Theorem 5.4).

2. Consider the extension of *While* with a binary left-first interleaving operator $\triangleleft$ specified by the rules

$$\frac{s, p \to s', p'}{s, p \triangleleft q \to s', q \triangleleft p'} \qquad \frac{s, p \downarrow s'}{s, p \triangleleft q \to s', q}$$

Again, $\simeq$ is not a congruence: For $t_1 = (\mathtt{x} := 2\mathtt{;}\ \mathtt{x} := \mathtt{x} + 2)$ and $t_2 = (\mathtt{x} := 2\mathtt{;}\ \mathtt{x} := \mathtt{x} * 2)$, we have $t_1 \simeq t_2$ but $t_1 \triangleleft (\mathtt{x} := 0) \not\simeq t_2 \triangleleft (\mathtt{x} := 0)$. Indeed the left of the above rules is receiving but the target of its conclusion does not have one of the allowed forms.

3. Extend *While* with a step-by-step branching operator $\triangledown$ specified by

$$\frac{s, p \to s_1, p' \qquad s, q \to s_2, q'}{s, p \triangledown q \to s_1, p' \triangledown q} P(s) \qquad \frac{s, p \to s_1, p' \qquad s, q \to s_2, q'}{s, p \triangledown q \to s_2, p \triangledown q'} \neg P(s)$$

and termination in all other cases. If the predicate $P \subseteq S$ is, for example, $\mathtt{x} = 0$, then the same t_1, t_2 as in item 2 witness that $\simeq$ is not a congruence: We have $t_1 \simeq t_2$ but $t_1 \triangledown (\mathtt{x} := 0) \not\simeq t_2 \triangledown (\mathtt{x} := 0)$. In this case, the condition that is violated is the requirement that all rules for $\triangledown$ must be receiving in the same position.

4. Consider the operator $\lceil\cdot\rceil$ specified by

$$\frac{s, p \to s', p'}{s, \lceil p \rceil \to s', \lceil p' \rceil} \qquad \frac{s, p \downarrow s'}{s, \lceil p \rceil \to s', p}$$

Again, t_1, t_2 as in item 2 witness failure of congruence: $t_1 \simeq t_2$ but $\lceil t_1 \rceil \not\simeq \lceil t_2 \rceil$. Indeed, the second rule violates Definition 5.2 as p terminates but is not discarded.

5.2 Cool stateful SOS

We now further restrict the streamlined format as follows:

▶ **Definition 5.8.** A stateful SOS specification is *cool* if for every active operator f there exists $j \in \{1, \dots, n\}$ (again called the *receiving position of* f) such that the following holds:

1. All rules for f whose j-th premiss is progressing are of the form

$$\frac{s, x_j \to s', y_j}{s, \mathsf{f}(x_1, \dots, x_n) \to s', \mathsf{f}(x_1, \dots, x_n)[y_j/x_j]}$$

2. All rules for f whose j-th premiss is terminating are of the form

$$\frac{s, x_j \downarrow s'}{s, \mathsf{f}(x_1, \dots, x_n) \to s'', t} \qquad \text{where } t \in \Sigma^\star(\{x_1, \dots, x_n\} \smallsetminus \{x_j\}) \text{ or } t = *,$$

and moreover s'' and t depend only on s' but not on s.

A stateful SOS specification is *uncool* if it is not cool.

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. In GSOS, rules of type 1 (without states) are known as *patience rules* [40].

▶ **Example 5.9.** The rules of the *While* language, which we have already observed to be streamlined (Example 5.3), are also cool.

Cool stateful SOS specifications are streamlined, and all of the negative examples from Section 5.1 apply here as well. Here is an example that separates the two concepts:

▶ **Example 5.10.** The sequential composition semantics with interrupts from Example 5.6 is uncool, as the third rule has a progressing premiss but is not of the form in Definition 5.8.1. Indeed, $\approx$ is not a congruence: For the predicate $\mathtt{x} = 42$ and the programs $t_1 = (\mathtt{x}{:=}42\mathtt{;}\ \mathtt{x}{:=}2)$ and $t_2 = (\mathtt{x} := 2)$, we have $t_1 \approx t_2$ but $t_1\mathtt{;}\ \mathtt{skip} \not\approx t_2\mathtt{;}\ \mathtt{skip}$.

As indicated above, coolness guarantees congruence for termination semantics:

▶ **Theorem 5.11.** *Termination semantics is compositional for cool stateful SOS specifications.*

Proof sketch. Suppose that $\mathsf{f} \in \Sigma$ is an n-ary operator and $p_m, q_m \in \mu\Sigma$ are programs with $p_m \approx q_m$ for $m = 1, \dots, n$. By symmetry, it suffices to show the following for all $s, \overline{s} \in S$:

> If $s, \mathsf{f}(p_1, \dots, p_m)$ terminates in state $\overline{s}$, then $s, \mathsf{f}(q_1, \dots, q_m)$ terminates in state $\overline{s}$.

The proof proceeds by an outer induction on the number of steps until termination of $s, \mathsf{f}(p_1, \dots, p_m)$ and an inner induction on the structure of the programs. ◀

By Corollary 5.5 we know that for every cool (whence streamlined) specification the kernel of $\mathsf{trc}\colon \nu T \nrightarrow (\nu B)^S$ forms a congruence. Since trc is surjective, this means precisely that there is a (unique) Σ-algebra structure on $(\nu B)^S$ for which trc is a Σ-algebra homomorphism.

▶ **Corollary 5.12.** *For every cool stateful SOS specification, the kernel of* $\mathsf{fn}^S\colon (\nu B)^S \nrightarrow (S+1)^S$ *is a congruence w.r.t. the induced* Σ*-algebra structure on* $(\nu B)^S$.

6 Conclusions and Future Work

We have introduced the *stateful SOS* rule format for the operational semantics of stateful languages, and equipped it with three semantics: resumption semantics, trace semantics, and termination semantics, in decreasing order of granularity. Our main interest has been in compositionality of these semantics. While resumption semantics is always compositional, it is in general undecidable whether the coarser semantics are compositional. However, compositionality is ensured by restricting to *streamlined* stateful SOS specifications for trace semantics, and to *cool* stateful SOS specifications for termination semantics. The compositionality result for the cool format improves on previous results for the similar *evaluation-in-context* formats [2] by abstracting from steps until termination. The streamlined format is more permissive, as we illustrate on a signal handling construct.

Our results currently work with deterministic state transformers, captured by the functor $TX = (BX)^S$ where $BX = S \times (X + 1)$. We believe that our results generalize to functors B equipped with a natural transformation $c_X\colon BX \to S$. As a first step, this generalization requires an abstract characterization of our streamlined and cool rule formats in terms of their corresponding natural transformations, along with categorical proofs of the respective congruence theorems. We leave this as an important point for future work.

A further direction of possible generalization is to cover *effects*, such as non-determinism, in a similar style as in work on evaluation-in-context [2]. Our work embeds the standard semantics of sequential imperative programming (in particular termination semantics) into the paradigm of operational semantics via distributive laws, and we expect to relate our results to work on morphisms of distributive laws [41, 21], which, for instance, have recently been shown to have applications to secure compilation [36]. Extending the overall paradigm to support higher-order languages is a well-known and, so far, elusive problem. Like in

the current work, tackling this problem may require a slight deviation from the standard form of GSOS laws. It is worth noting that rule formats for higher-order languages have been proposed in the past by Howe [16], Bernstein [7] and more recently Hirschowitz and Lafont [15].

Our treatment of resumption and trace semantics and their relationship is generic, and presumably can be transferred to other settings, in particular to constructive and type-theoretic frameworks. Indeed we expect that it can be implemented relatively directly in foundational proof assistants such as Agda, without additional postulates (such as the axiom of choice or the law of excluded middle). In contrast, the domain $(S+1)^S$ of termination semantics is inherently classical, as it postulates that every computation will either terminate or diverge. This can be remedied by replacing the maybe-monad $(-)+1$ with a suitable *partiality monad* [4, 12]. We will explore to what extent our results regarding termination semantics can be rebased on this more general perspective.

References

1 Faris Abou-Saleh and Dirk Pattinson. Towards effects in mathematical operational semantics. In Michael W. Mislove and Joël Ouaknine, editors, *Mathematical Foundations of Programming Semantics, MFPS 2011*, volume 276 of *Electron. Notes Theor. Comput. Sci.*, pages 81–104. Elsevier, 2011. doi:10.1016/j.entcs.2011.09.016.

2 Faris Abou-Saleh and Dirk Pattinson. Comodels and effects in mathematical operational semantics. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7794 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2013. doi:10.1007/978-3-642-37075-5_9.

3 Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 15(4):589–602, 1974.

4 Thorsten Altenkirch, Nils Danielsson, and Nicolai Kraus. Partiality, revisited - the partiality monad as a quotient inductive-inductive type. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *Lecture Notes Comput. Sci.*, pages 534–549, 2017.

5 Michael Barr. Coequalizers and free triples. *Math. Z.*, 116:307–322, 1970.

6 Falk Bartels. *On generalised coinduction and probabilistic specification formats: Distributive laws in coalgebraic modelling.* PhD thesis, Vrije Universiteit Amsterdam, 2004.

7 Karen L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *Thirteenth Annual IEEE Symposium on Logic in Computer Science, Indianapolis, Indiana, USA, June 21-24, 1998*, pages 153–164. IEEE Computer Society, 1998. doi:10.1109/LICS.1998.705652.

8 Bard Bloom. Structural operational semantics for weak bisimulations. *Theor. Comput. Sci.*, 146(1&2):25–68, 1995. doi:10.1016/0304-3975(94)00152-9.

9 Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995. doi:10.1145/200836.200876.

10 Bard Bloom and Frits Vandraager. Sos rule formats for parameterized and state-bearing processes. http://www.sws.cs.ru.nl/publications/papers/fvaan/bardfrits.ps, 1994.

11 Stephen D. Brookes. Full abstraction for a shared variable parallel language. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 98–109. IEEE Computer Society, 1993. doi:10.1109/LICS.1993.287596.

12 James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science*, 29(1):67–92, 2019.

13 Marcelo P. Fiore and Sam Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 49–58. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.7.

14 Matthew Hennessy and Gordon D. Plotkin. Full abstraction for a simple parallel programming language. In Jirí Becvár, editor, *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 1979. doi:10.1007/3-540-09526-8_8.

15 Tom Hirschowitz and Ambroise Lafont. A categorical framework for congruence of applicative bisimilarity in higher-order languages. *CoRR*, abs/2103.16833, 2021. URL: https://arxiv.org/abs/2103.16833, arXiv:2103.16833.

16 Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996. doi:10.1006/inco.1996.0008.

17 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.

18 Alan Jeffrey and Julian Rathke. Java Jr: Fully abstract trace semantics for a core Java language. In Shmuel Sagiv, editor, *14th European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 423–438. Springer, 2005. doi:10.1007/978-3-540-31987-0_29.

19 Gilles Kahn. Natural semantics. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS 87, 4th Annual Symposium on Theoretical Aspects of Computer Science, Passau, Germany, February 19-21, 1987, Proceedings*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 1987. doi:10.1007/BFb0039592.

20 Bartek Klin. Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.*, 412(38):5043–5069, 2011. doi:10.1016/j.tcs.2011.03.023.

21 Bartek Klin and Beata Nachyla. Presenting morphisms of distributive laws. In *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands*, pages 190–204, 2015. doi:10.4230/LIPIcs.CALCO.2015.190.

22 Bartek Klin and Vladimiro Sassone. Structural operational semantics for stochastic process calculi. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2008. doi:10.1007/978-3-540-78499-9_30.

23 Xavier Leroy and Hervé Grall. Coinductive big-step operational semantics. *CoRR*, abs/0808.0586, 2008. arXiv:0808.0586.

24 Marino Miculan and Marco Peressotti. Structural operational semantics for non-deterministic processes with quantitative aspects. *Theor. Comput. Sci.*, 655:135–154, 2016. doi:10.1016/j.tcs.2016.01.012.

25 Mohammad Reza Mousavi, Michel Reniers, and Jan Friso Groote. Congruence for sos with data. In *LICS*, pages 302–313. IEEE Computer Society Press, 2004.

26 Keiko Nakata and Tarmo Uustalu. Trace-based coinductive operational semantics for while. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2009. doi:10.1007/978-3-642-03359-9_26.

27 Scott Owens, Magnus O. Myreen, Ramana Kumar, and Yong Kiam Tan. Functional big-step semantics. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016,*

Proceedings, volume 9632 of *Lecture Notes in Computer Science*, pages 589–615. Springer, 2016. doi:10.1007/978-3-662-49498-1_23.

28 Marco Patrignani and Dave Clarke. Fully abstract trace semantics for protected module architectures. *Comput. Lang. Syst. Struct.*, 42:22–45, 2015. doi:10.1016/j.cl.2015.03.002.

29 Marco Patrignani, Dominique Devriese, and Frank Piessens. On modular and fully-abstract compilation. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 17–30. IEEE Computer Society, 2016. doi:10.1109/CSF.2016.9.

30 Maciej Piróg and Jeremy Gibbons. Monads for behaviour. In *Mathematical Foundations of Programming Semantics, MFPS 2013*, volume 298 of *Electron. Notes Theor. Comput. Sci.*, pages 309–324, 2015.

31 Andrew M. Pitts. Operational semantics and program equivalence. In Gilles Barthe, Peter Dybjer, Luís Pinto, and João Saraiva, editors, *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science*, pages 378–412. Springer, 2000. doi:10.1007/3-540-45699-6_8.

32 Andrew M. Pitts and Ian D. B. Stark. Operational reasoning for functions with local state. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–274. Cambridge University Press, New York, NY, USA, 1998.

33 Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

34 Jan J. M. M. Rutten. A note on coinduction and weak bisimilarity for while programs. *ITA*, 33(4/5):393–400, 1999. doi:10.1051/ita:1999125.

35 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3 – 80, 2000.

36 Stelios Tsampas, Andreas Nuyts, Dominique Devriese, and Frank Piessens. A categorical approach to secure compilation. In Daniela Petrisan and Jurriaan Rot, editors, *Coalgebraic Methods in Computer Science - 15th IFIP WG 1.3 International Workshop, CMCS 2020, Colocated with ETAPS 2020, Dublin, Ireland, April 25-26, 2020, Proceedings*, volume 12094 of *Lecture Notes in Computer Science*, pages 155–179. Springer, 2020. doi:10.1007/978-3-030-57201-3_9.

37 Stelios Tsampas, Christian Williams, Andreas Nuyts, Dominique Devriese, and Frank Piessens. Abstract congruence criteria for weak bisimilarity. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 88:1–88:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.88.

38 Daniele Turi. Categorical modelling of structural operational rules: Case studies. In *Category Theory and Computer Science, 7th International Conference, CTCS '97, Santa Margherita Ligure, Italy, September 4-6, 1997, Proceedings*, pages 127–146, 1997. doi:10.1007/BFb0026985.

39 Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 280–291, 1997. doi:10.1109/LICS.1997.614955.

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

41 Hiroshi Watanabe. Well-behaved translations between structural operational semantics. *Electr. Notes Theor. Comput. Sci.*, 65(1):337–357, 2002. doi:10.1016/S1571-0661(04)80372-4.

A Appendix: Omitted Proofs

Proof of Theorem 3.9

Let $B = S \times (\mathrm{Id} + 1)$. The correspondence between (2) and (3) is proven by the following bijections:

$$
\begin{aligned}
&\mathsf{Nat}(S \times \Sigma(\mathrm{Id} \times B), B\Sigma^\star) \\
&\cong \mathsf{Nat}\Big(\textstyle\sum_{\mathsf{f}\in\Sigma} S \times (\mathrm{Id} \times B)^{\mathsf{ar}(\mathsf{f})}, B\Sigma^\star\Big) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{Nat}(S \times (\mathrm{Id} \times B)^{\mathsf{ar}(\mathsf{f})}, B\Sigma^\star) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{Nat}(S \times (\mathrm{Id} \times S \times (\mathrm{Id} + 1))^{\mathsf{ar}(\mathsf{f})}, B\Sigma^\star) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{Nat}(S \times S^{\mathsf{ar}(\mathsf{f})} \times \mathrm{Id}^{\mathsf{ar}(\mathsf{f})} \times (\mathrm{Id} + 1)^{\mathsf{ar}(\mathsf{f})}, B\Sigma^\star) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{Nat}(\mathrm{Id}^{\mathsf{ar}(\mathsf{f})} \times (\mathrm{Id} + 1)^{\mathsf{ar}(\mathsf{f})}, (B\Sigma^\star)^{S\times S^{\mathsf{ar}(\mathsf{f})}}) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{Nat}\Big(\mathrm{Id}^{\mathsf{ar}(\mathsf{f})} \times \sum_{W\subseteq\mathsf{ar}(\mathsf{f})} \mathrm{Id}^{W}, (B\Sigma^\star)^{S\times S^{\mathsf{ar}(\mathsf{f})}}\Big) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \prod_{W\subseteq\mathsf{ar}(\mathsf{f})} \mathsf{Nat}(\mathrm{Id}^{\mathsf{ar}(\mathsf{f})} \times \mathrm{Id}^{W}, (B\Sigma^\star)^{S\times S^{\mathsf{ar}(\mathsf{f})}}) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \prod_{W\subseteq\mathsf{ar}(\mathsf{f})} \mathsf{Nat}(\mathrm{Id}^{\mathsf{ar}(\mathsf{f})+W}, (B\Sigma^\star)^{S\times S^{\mathsf{ar}(\mathsf{f})}}) \\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \prod_{W\subseteq\mathsf{ar}(\mathsf{f})} (B\Sigma^\star(\mathsf{ar}(\mathsf{f}) + W))^{S\times S^{\mathsf{ar}(\mathsf{f})}}
\end{aligned}
$$

The last step uses the Yoneda lemma, and all other steps follow from the definition of (co-)products and the fact that products distribute over coproducts in **Set**.

To prove the correspondence between (1) and (2) we first observe that every stateful SOS specification $\mathcal{L}$ induces a family

$$\big(r_{\mathsf{f},W} \colon S \times S^{\mathsf{ar}(\mathsf{f})} \to S \times \Sigma^\star(\mathsf{ar}(\mathsf{f}) + W) + S\big)_{\mathsf{f}\in\Sigma, W\subseteq\mathsf{ar}(\mathsf{f})}$$

as follows: for every n-ary $\mathsf{f} \in \Sigma$, $s, s'_1, \dots, s'_n \in S$ and $W \subseteq \{1, \dots, n\}$, if $\mathcal{L}$ contains the rule

$$\frac{(s, x_j \to s'_j, y_j)_{j\in W} \qquad (x_j \downarrow s'_j)_{j\in\{1,\dots,n\}\setminus W}}{s, \mathsf{f}(x_1, \dots, x_n) \to s', t},$$

then $r_{\mathsf{f},W}(s, s'_1, \dots, s'_n) = (s', t')$ where $t' \in \Sigma^\star(\mathsf{ar}(\mathsf{f}) + W)$ is obtained from t by substituting the variable x_j by j (in the left coproduct component of $\mathsf{ar}(\mathsf{f}) + W$) for each $j \in \mathsf{ar}(\mathsf{f})$, and the variable y_j by j (in the right coproduct component of $\mathsf{ar}(\mathsf{f}) + W$) for each $j \in W$. If $\mathcal{L}$ contains the rule

$$\frac{(s, x_j \to s'_j, y_j)_{j\in W} \qquad (x_j \downarrow s'_j)_{j\in\{1,\dots,n\}\setminus W}}{s, \mathsf{f}(x_1, \dots, x_n) \downarrow s'}.$$

then $r_{\mathsf{f},W}(s, s'_1, \dots, s'_n) = s'$. This translation clearly defines a bijective correspondence between stateful SOS specifications $\mathcal{L}$ and families $(r_{\mathsf{f},W})_{\mathsf{f},W}$.

The construction $\mathcal{L} \mapsto \delta$ in Remark 3.8.1 is obtained by first forming the family $(r_{\mathsf{f},W})_{\mathsf{f},W}$ corresponding to $\mathcal{L}$ and then taking the stateful SOS law δ corresponding to $(r_{\mathsf{f},W})_{\mathsf{f},W}$ according to the above isomorphism.

Similarly, the construction $\delta \mapsto \mathcal{L}$ in Remark 3.8.2 is obtained by first forming the family $(r_{\mathsf{f},W})_{\mathsf{f},W}$ corresponding to δ according to the above isomorphism, and then taking the stateful SOS specification $\mathcal{L}$ corresponding to $(r_{\mathsf{f},W})_{\mathsf{f},W}$.

Consequently, the two constructions in Remark 3.8 are mutually inverse. ◀

Proof of Proposition 4.4

▶ **Remark A.1.** The proof of Proposition 4.4 makes use of an induction principle which combines two induction principles known in the literature: *primitive recursion* (see e.g. [17, Prop. 2.4.7]) and *induction with parameters* (see e.g. [17, Exercise. 2.5.5]. We recall primitive recursion in item 1 below and use it in item 2 to establish the combined principle we need.

1. Let F be a functor with an initial algebra μF on a category with finite products. Primitive recursion states that for every morphism $\alpha\colon F(\mu F \times A) \to A$ there exists a unique morphism $h\colon \mu F \to A$ such that the following square commutes:

$$\begin{array}{ccc} F(\mu F) & \xrightarrow{\iota} & \mu F \\ {\scriptstyle F\langle \mathsf{id}, h\rangle}\downarrow & & \downarrow{\scriptstyle h} \\ F(\mu F \times A) & \xrightarrow{\alpha} & A \end{array}$$

2. Now assume in addition that F is strong and that the base category is cartesian closed. Then for every object Y and every morphism $\beta\colon Y \times F(\mu F \times A) \to A$ there exists a unique morphism $h\colon Y \times \mu F \to A$ such that the following diagram commutes:

$$\begin{array}{ccc} Y \times F(\mu F) & \xrightarrow{\mathsf{id}\times\iota} & Y \times \mu F \\ {\scriptstyle \langle \mathsf{fst},\mathsf{st}\rangle}\downarrow & & \\ Y \times F(Y \times \mu F) & & \downarrow{\scriptstyle h} \\ {\scriptstyle \mathsf{id}\times F\langle \mathsf{snd}, h\rangle}\downarrow & & \\ Y \times F(\mu F \times A) & \xrightarrow{\beta} & A \end{array} \tag{A.1}$$

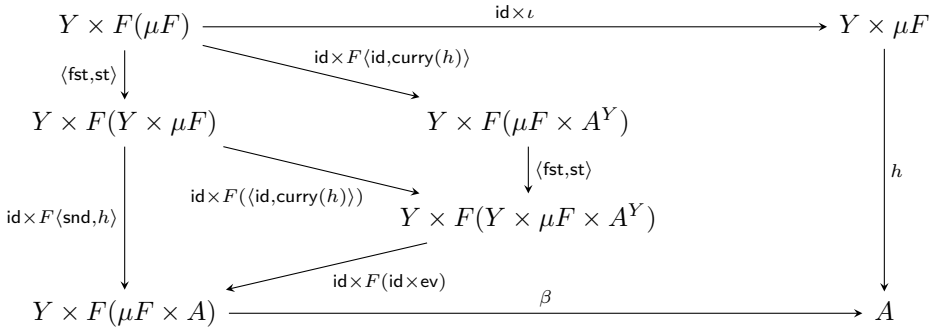
Indeed, given β one obtains a morphism $\alpha\colon F(\mu F \times A^Y) \to A^Y$ by currying the morphism

$$\begin{aligned} Y \times F(\mu F \times A^Y) &\xrightarrow{\langle \mathsf{fst},\mathsf{st}\rangle} Y \times F(Y \times \mu F \times A^Y) \cong Y \times F(\mu F \times Y \times A^Y) \\ &\xrightarrow{\mathsf{id}\times F(\mathsf{id}\times \mathsf{ev})} Y \times F(\mu F \times A) \xrightarrow{\beta} A. \end{aligned} \tag{A.2}$$

By primitive recursion there exists a unique morphism $h\colon Y \times \mu F \to A$ such that the square below commutes:

$$\begin{array}{ccc} F(\mu F) & \xrightarrow{\iota} & \mu F \\ {\scriptstyle F\langle \mathsf{id}, \mathsf{curry}(h)\rangle}\downarrow & & \downarrow{\scriptstyle \mathsf{curry}(h)} \\ F(\mu F \times A^Y) & \xrightarrow{\alpha} & A^Y \end{array} \tag{A.3}$$

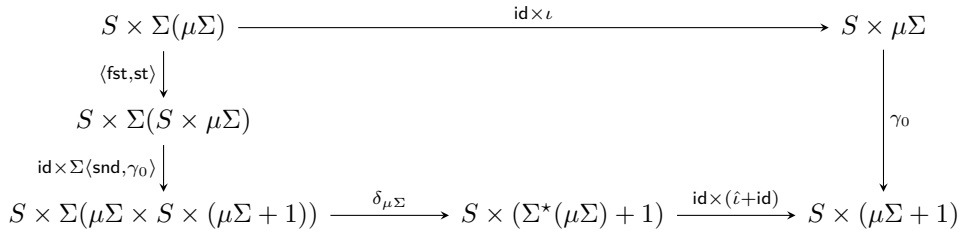
Now consider the diagram below:



(A.4)

Its outside is (A.1), the right-hand part is precisely the uncurrying of (A.3), and the remaining two inner parts clearly commute. Hence, the outside commutes iff so does the right-hand part which happens iff (A.3) commutes. This proves the desired result.

Proof of Proposition 4.4. In Definition 4.3, the map $\gamma_0 \colon S \times \mu\Sigma \to S \times (\mu\Sigma + 1)$ is defined such that the following diagram commutes:



This is an instance of (A.1) with $h = \gamma_0$ and $\beta = (\mathsf{id} \times (\hat{\iota} + \mathsf{id})) \cdot \delta_{\mu\Sigma}$. The map α in (A.3) is the $\mu\Sigma$-component of the natural transformation in (4.4) composed with $\mathsf{id} \times (\hat{\iota} + \mathsf{id})$. Thus currying this yields $T\hat{\iota} \cdot \hat{\delta}_{\mu\Sigma}$ (cf. (4.1)). This implies the desired result. ◀

Proof of Proposition 4.7

The object $(\nu B)^S$ can be equipped with the T-coalgebra structure

$$(\nu B)^S \xrightarrow{\beta^S} (B(\nu B))^S = T(\nu B) \xrightarrow{T\Delta} T((\nu B)^S),$$

where β is the structure of the final coalgebra νB and $\Delta = \langle \mathsf{id} \rangle_{s \in S}$ denotes the diagonal. We claim that the unique T-coalgebra homomorphism

$$m = (T\Delta \cdot \beta^S)^\sharp \colon (\nu B)^S \to \nu T$$

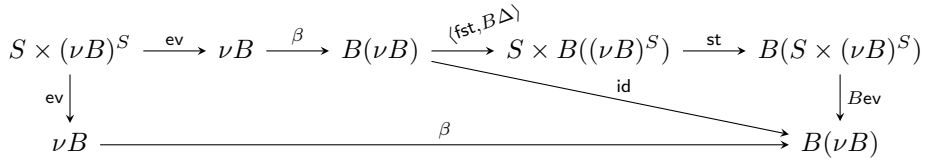
is a splitting of trc (that is, $\mathsf{trc} \cdot m = \mathsf{id}$), which implies that trc is surjective. To prove the claim, we consider the following B-coalgebra structure on $S \times (\nu B)^S$:

$$S \times (\nu B)^S \xrightarrow{\mathsf{ev}} \nu B \xrightarrow{\beta} B(\nu B) \xrightarrow{\langle \mathsf{fst}, B\Delta\rangle} S \times B((\nu B)^S) \xrightarrow{\mathsf{st}} B(S \times (\nu B)^S).$$

We will show below that both $\mathsf{ev} \colon S \times (\nu B)^S \to \nu B$ and $\bar{\tau}^\sharp \cdot (S \times m)$ are homomorphisms from this coalgebra to νB. Then $\bar{\tau}^\sharp \cdot (S \times m) = \mathsf{ev}$ by finality, whence the desired result holds as follows:

$$\mathsf{trc} \cdot m = \mathsf{curry}(\bar{\tau}^\sharp) \cdot m = \mathsf{curry}(\mathsf{ev}) = \mathsf{id}.$$

The commutative diagram below proves ev to be a coalgebra homomorphism:



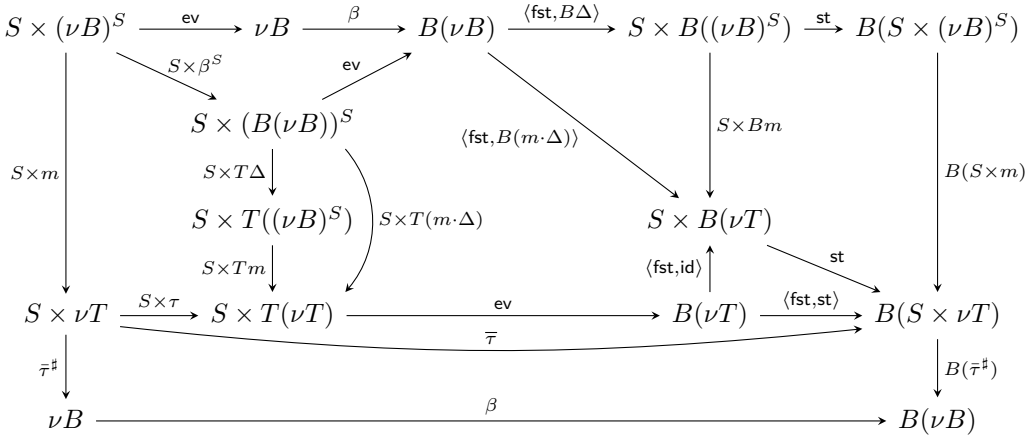
To see that the right-hand triangle commutes, let $(s, w) \in S \times (\nu B + 1) = B(\nu B)$. If $w \in \nu B$, then we have

$$(s, w) \xmapsto{\langle \mathsf{fst}, B\Delta\rangle} (s, (s, c_w)) \xmapsto{\mathsf{st}} (s, (s, c_w)) \xmapsto{B\mathsf{ev}} (s, w)$$

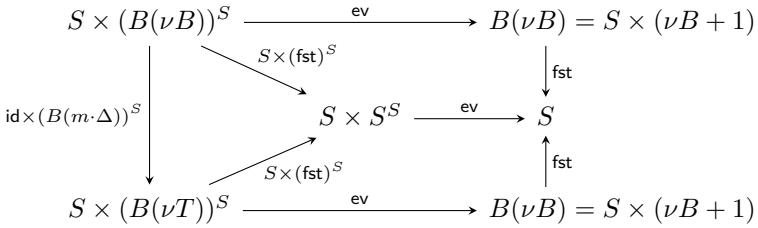
where $c_w \in (\nu B)^S$ is the constant map with value w. If $w = *$, we have

$$(s,w) = (s,*) \xmapsto{\langle \mathsf{fst}, B\Delta\rangle} (s,(s,*)) \xmapsto{\mathsf{st}} (s,*) \xmapsto{B\mathsf{ev}} (s,*) = (s,w).$$

Similarly, the diagram below shows that $\bar{\tau}^\sharp \cdot (S \times m)$ is a coalgebra homomorphism:



Indeed, we show that all its inner parts commutes, whence so does the outside as desired. The lowest part commutes since $\overline{\tau}^\sharp$ is a coalgebra homomorphism, and the small part above it is the definition of $\overline{\tau}$. The upper right-hand part commutes by the naturality of $\mathsf{st}\colon S \times BX \to B(S \times X)$, and the triangles below it and on its left obviously do. The upper left-hand triangle commutes by the naturality of ev, and the left-hand part does since $m\colon (\nu B)^S \to \nu T$ is a T-coalgebra homomorphism. For the remaining middle part we consider the components of the product $S \times B(\nu T)$ separately. The right-hand component commutes due to the naturality of $\mathsf{ev}\colon S \times TX = S \times (BX)^S \to BX$, and for the left-hand component we expand the definition of T and obtain the following commutative diagram:



The two right-hand parts obviously commute, and the left-hand triangle also clearly does: remove $S \times (\text{–})^S$ and use that $B(m \cdot \Delta) = S \times (m \cdot \Delta + 1)$. This completes the proof. ◀

Proof of Theorem 4.12

We split Theorem 4.12 into two separate results:

▶ **Theorem A.2.** *It is undecidable whether the trace semantics induced by a primitive recursive stateful SOS specification is compositional.*

Proof. We reduce the undecidable problem whether a given Turing machine M halts on the empty tape to the compositionality problem. Let Q be the set of states and $A \cup \{\sharp\}$ the tape alphabet with $\sharp$ representing empty cells. Recall that a *configuration* of M is a pair $C = (q, v\underline{a}w)$ where $q \in Q$ is the current state, vaw is the tape content and the underscore

indicates the head position. We write $C \vdash C'$ if C' is the successor configuration of C according to the transitions of M. The initial configuration is $C_0 = (q_0, \underline{\sharp})$, and we assume w.l.o.g. that there is a unique halting configuration C_{halt}.

From M we construct a stateful SOS specification $\mathcal{L}$ as follows. The signature is $\Sigma = \{i, j, c_0, c, d_0, d, u\}$ with u unary and all other symbols constants, and the states are

$$S = \mathrm{Conf} \cup \{0, 1, \mathsf{err}\} \cup \mathbb{N} \times \{c_0, c, d_0, d\}$$

where Conf is the set of configurations of M. The rules are listed below; all cases where no explicit rule is specified lead to termination in the error state err.

- Rules for i and j:

  $$\frac{}{s, i \to C_0, c_0} \qquad \frac{}{s, j \to C_0, d_0}$$

  for all $s \in S$.
- Rules for c_0 and d_0:

  $$\frac{}{C, c_0 \to C', c_0} \qquad \frac{}{C_{\mathrm{halt}}, c_0 \to 0, c} \qquad \frac{}{(n, c_0), c_0 \downarrow (n, c_0)}$$

  $$\frac{}{C, d_0 \to C', d_0} \qquad \frac{}{C_{\mathrm{halt}}, d_0 \to 0, d} \qquad \frac{}{(n, d_0), d_0 \downarrow (n, d_0)}$$

  for all $C, C' \in \mathrm{Conf}$ with $C \vdash C'$ and $n \in \mathbb{N}$.
- Rules for c and d:

  $$\frac{}{0, c \to 0, c} \qquad \frac{}{1, c \downarrow 1} \qquad \frac{}{(n, c), c \downarrow (n, c)}$$

  $$\frac{}{0, d \to 0, d} \qquad \frac{}{1, d \downarrow 0} \qquad \frac{}{(n, d), d \downarrow (n, d)}$$

  for all $n \in \mathbb{N}$.
- Rules for u:

  $$\frac{s, p \to s', p'}{s, u(p) \to 1, p'} \qquad \frac{C, p \to s', p'}{C, u(p) \to s', u(p')} \qquad \frac{}{(n, k), u(p) \to (n+1, k), p}$$

  for all $s \in \{0, 1\}$, $s' \in S$, $C \in \mathrm{Conf}$ and $(n, k) \in \mathbb{N} \times \{c_0, c, d_0, d\}$.

Clearly $\mathcal{L}$ has a primitive recursive representation derivable from the description of the given machine M. For the induced trace equivalence $\simeq = \simeq_{\mathcal{L}}$ we now show that

$$M \text{ halts on the empty tape} \qquad \text{iff} \qquad \simeq \text{ is not a congruence.}$$

From the undecidability of the halting problem on the empty tape it then follows that the congruence property of $\simeq$, i.e. compositionality of $\mathcal{L}$ under trace semantics, is undecidable.

($\Longrightarrow$) Suppose that M halts on the empty tape, and let $C_0 \vdash C_1 \vdash \ldots \vdash C_n = C_{\mathrm{halt}}$ be the finite run of M. Then $i \simeq j$ but $u(i) \not\simeq u(j)$. Indeed, running i and j on any input s yields

$$s, i \to C_0, c_0 \to C_1, c_0 \to \cdots \to C_n = C_{\mathrm{halt}}, c_0 \to 0, c \to 0, c \to \cdots$$

and

$$s, j \to C_0, d_0 \to C_1, d_0 \to \cdots \to C_n = C_{\mathrm{halt}}, d_0 \to 0, d \to 0, d \to \cdots,$$

and so $i \simeq j$. On the other hand, running $u(i)$ and $u(j)$ on input C_0 yields

$$C_0, u(i) \to C_0, u(c_0) \to C_1, u(c_0) \to \cdots \to C_n = C_{\mathrm{halt}}, u(c_0) \to 0, u(c) \to 1, c \downarrow 1$$

and

$$C_0, u(j) \to C_0, u(d_0) \to C_1, u(d_0) \to \cdots \to C_n = C_{\mathrm{halt}}, u(d_0) \to 0, u(d) \to 1, d \downarrow 0,$$

whence $u(i) \not\simeq u(j)$. This shows that $\simeq$ is not a congruence.

($\Longleftarrow$) We argue by contraposition. Suppose that M does not halt on the empty tape, and let $C_0 \vdash C_1 \vdash C_2 \vdash \cdots$ be the infinite run of M. We claim that $\simeq$ is the finest equivalence relation containing the pairs

$$u^n(i) \simeq u^n(j) \qquad (n \in \mathbb{N})$$

(that is, $\simeq$ consists of these pairs, their converses, and all diagonal pairs). This equivalence relation is clearly a congruence. Thus let us prove the claim:

1. We have $i \simeq j$: running i and j on any input state s yields

   $$s, i \to C_0, c_0 \to C_1, c_0 \to C_2, c_0 \to \cdots \qquad \text{and} \qquad s, j \to C_0, d_0 \to C_1, d_0 \to C_2, d_0 \to \cdots.$$

   Thus i and j are trace equivalent.
2. We have $u^n(i) \simeq u^n(j)$ for all $n \geq 1$: running $u^n(i)$ on all possible input states yields the following computations, where $s \in \{0,1\}$, $C \in \mathrm{Conf}$, and $(m,k) \in \mathbb{N} \times \{c_0, c, d_0, d\}$:
   - $s, u^n(i) \to 1, c_0 \downarrow \mathsf{err}$
   - $C, u^n(i) \to C_0, u^n(c_0) \to C_1, u^n(c_0) \to C_2, u^n(c_0) \to \cdots$
   - $(m,k), u^n(i) \to (m{+}1,k), u^{n-1}(i) \to \cdots \to (m{+}n,k), i \to C_0, c_0 \to C_1, c_0 \to C_2, c_0 \to \cdots$
   - $\mathsf{err}, u^n(i) \downarrow \mathsf{err}$

   The computations for $u^n(j)$ are analogous, with d_0 in place of c_0. Thus $u^n(i)$ and $u^n(j)$ are trace equivalent.
3. We have $u^m(i) \not\simeq u^n(i)$ and $u^m(j) \not\simeq u^n(j)$ for $m \neq n$. To see this, suppose w.l.o.g. that $m < n$. Running $u^m(i)$ and $u^n(i)$ on input state $(0, c_0)$ yields:
   - $(0,c_0), u^m(i) \to (1,c_0), u^{m-1}(i) \to \cdots \to (m,c_0), i \to C_0, c_0 \to C_1, c_0 \to \cdots$
   - $(0,c_0), u^n(i) \to (1,c_0), u^{n-1}(i) \to \cdots \to (m,c_0), u^{n-m}(i) \to (m+1,c_0), u^{n-m-1}(i) \to \cdots$

   Since the two computations lead to different states after $m+1$ steps, we see that $u^m(i) \not\simeq u^n(i)$. The same argument with d_0 in place of c_0 shows $u^m(j) \not\simeq u^n(j)$.
4. For each $n \in \mathbb{N}$ the program $u^n(c)$ is not $\simeq$-equivalent to any other program. In fact, $u^n(c)$ is the unique program that on input state $(0,c)$ eventually terminates in the state (n,c). Analogously, none of the programs $u^n(c_0)$, $u^n(d_0)$ and $u^n(d)$ is $\simeq$-equivalent to any other program. ◀

▶ **Theorem A.3.** *It is undecidable whether the termination semantics induced by a primitive recursive stateful SOS specification is compositional.*

Proof. We use the same reduction $M \mapsto \mathcal{L}$ as in the proof of Theorem A.2. For the equivalence relation $\approx = \approx_{\mathcal{L}}$ we show that

$$M \text{ halts on the empty tape} \qquad \text{iff} \qquad \approx \text{ is not a congruence.}$$

The proof of ($\Longrightarrow$) is as before (with $\simeq$ replaced by $\approx$). In the proof of ($\Longleftarrow$) observe that the equivalence relation $\approx$ is now generated by the pairs

$$i \approx j \qquad \text{and} \quad u^m(i) \approx u^n(j) \qquad (m, n \geq 1).$$

This is immediate from the analysis of computations in items 1, 2 and 4 of the proof of Theorem A.2. ◀

Proof of Theorem 5.4

▶ **Remark A.4.** Recall the (dual of) the classical result by Adámek [3] that for every category $\mathcal{C}$ with a terminal object 1 and limits of ω^{op}-chains and every endofunctor B on $\mathcal{C}$ preserving such limits, the final coalgebra νB is obtained as the limit of the ω^{op}-chain

$$1 \xleftarrow{!} B1 \xleftarrow{B!} B^2 1 \xleftarrow{B^2!} \cdots \xleftarrow{B^{k-1}!} B^k 1 \xleftarrow{B^k!} B^{k+1}1 \xleftarrow{B^{k+1}!} \cdots$$

where ! is the unique morphism and B^k means k-fold application of B. For the functor $BX = S \times (X+1)$ on **Set**, letting $S^{\leq k} \subseteq S^+$ denote the set of nonempty finite S-streams of length at most k, we have $B^k 1 \cong S^{\leq k} + S^k$ and $\nu B \cong S^+ + S^\omega$. The limit projection $\pi_k \colon \nu B \to B^k 1$ maps every string $w \in S^+$ to its prefix of length at most k in the first coproduct component and every stream in S^ω to its prefix of length k in the second coproduct component.

▶ **Notation A.5.** For $k \in \mathbb{N}$, we put

$$[\![-]\!]_k = (\mu\Sigma \xrightarrow{[\![-]\!]} (\nu B)^S \xrightarrow{\pi_k^S} (B^k 1)^S).$$

Thus, given $p \in \mu\Sigma$ and $s \in S$, the stream $[\![p]\!]_k(s) \in B^k 1 = S^{\leq k} + S^k$ is the k-step behaviour of the program p on the input state s; note that $[\![p]\!]_k$ retains the information whether p has terminated after at most k steps or not. For $p, q \in \mu\Sigma$ and $s \in S$, we put

$$p \simeq_k q \quad \text{iff} \quad [\![p]\!]_k = [\![q]\!]_k \qquad \text{and} \qquad p \simeq_{k,s} q \quad \text{iff} \quad [\![p]\!]_k(s) = [\![q]\!]_k(s).$$

Observe that

$$p \simeq q \qquad \text{iff} \qquad p \simeq_k q \quad \text{for all } k.$$

Proof of Theorem 5.4. To prove that $\simeq$ is a congruence, it suffices to prove that each $\simeq_k$ is a congruence. To this end, we shall establish the following slightly stronger statement:

Claim. For every $k \in \mathbb{N}$, every n-ary operator $\mathsf{f} \in \Sigma$, and $p_m, q_m \in \mu\Sigma$ for $m = 1, \ldots, n$,

1. if f is passive, then

$$p_m \simeq_k q_m \quad (m = 1, \ldots, n) \qquad \Longrightarrow \qquad \mathsf{f}(p_1, \ldots, p_n) \simeq_k \mathsf{f}(q_1, \ldots, q_n),$$

2. if f is active with receiving position $j \in \{1, \ldots, n\}$ and $s \in S$, then

$$p_m \simeq_k q_m \ (m \in \{1, \ldots, n\} \setminus \{j\}) \quad \text{and} \quad p_j \simeq_{k,s} q_j \quad \Longrightarrow \quad \mathsf{f}(p_1, \ldots, p_n) \simeq_{k,s} \mathsf{f}(q_1, \ldots, q_n).$$

The proof of the claim is by induction on k. The induction base ($k = 0$) holds trivially because $p \simeq_{0,s} q$ and $p \simeq_0 q$ for all programs p, q and states s.

Now for the induction step $k \to k+1$. Further below we write : for the prefixing operation on finite lists.

1. Suppose that f is passive and that $p_m \simeq_{k+1} q_m$ for $m = 1, \ldots, n$. We need to show $\mathsf{f}(p_1, \ldots, p_n) \simeq_{k+1} \mathsf{f}(q_1, \ldots, q_n)$, that is,

$$[\![\mathsf{f}(p_1, \ldots, p_n)]\!]_{k+1}(s) = [\![\mathsf{f}(q_1, \ldots, q_n)]\!]_{k+1}(s) \qquad \text{for every } s \in S.$$

Given $s \in S$, the rule applying to s and f is of the form

$$\frac{}{s, \mathsf{f}(x_1, \ldots, x_n) \to s', t} \qquad \text{where } t \in \Sigma^\star(\{x_1, \ldots, x_n\}) \text{ or } t = *.$$

If $t = *$, then

$$[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) = s' = [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s).$$

Otherwise $t \in \Sigma^\star(\{x_1,\dots,x_n\})$ and

$$\begin{aligned}
[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) &= s' : [\![t(p_1,\dots,p_n)]\!]_k(s') && \text{def. of } [\![-]\!]\\
&= s' : [\![t(q_1,\dots,q_n)]\!]_k(s')\\
&= [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s) && \text{def. of } [\![-]\!],
\end{aligned}$$

where in the second step we use that $p_m \simeq_k q_m$ for $m = 1,\dots,n$ and that $\simeq_k$ is a congruence by induction.

2. Suppose that f is active with receiving position j. Let $s \in S$ and $p_m \simeq_{k+1} q_m$ for $m \in \{1,\dots,n\} \setminus \{j\}$ and $p_j \simeq_{k+1,s} q_j$. We need to show $\mathsf{f}(p_1,\dots,p_n) \simeq_{k+1,s} \mathsf{f}(q_1,\dots,q_n)$, that is,

$$[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) = [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s).$$

Let $s, p_j \to s', p$ and $s, q_j \to s', q$ be the first computation steps of p_j and q_j, respectively; since $p_j \sim_{1,s} q_j$, both steps lead to the same state s'. Suppose first that the rule applying to s and f is receiving and thus of the form

$$\frac{s, x_j \to s', y}{s, \mathsf{f}(x_1,\dots,x_n) \to s', t} \qquad \text{where } t = \{\mathsf{f}(x_1,\dots,x_n)[y/x_j]\} \text{ or } t = y.$$

If $t = \mathsf{f}(x_1,\dots,x_n)[y/x_j]$, then

$$\begin{aligned}
[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) &= s' : [\![\mathsf{f}(p_1,\dots,p_{j-1},p,p_{j+1},\dots,p_n)]\!]_k(s') && \text{def. of } [\![-]\!]\\
&= s' : [\![\mathsf{f}(q_1,\dots,q_{j-1},q,q_{j+1},\dots,q_n)]\!]_k(s')\\
&= [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s) && \text{def. of } [\![-]\!],
\end{aligned}$$

where in the second step we use that $p_m \simeq_k q_m$ for $m \neq j$ and $p \simeq_{k,s'} q$, which by induction implies

$$[\![\mathsf{f}(p_1,\dots,p_{j-1},p,p_{j+1},\dots,p_n)]\!]_k(s') = [\![\mathsf{f}(q_1,\dots,q_{j-1},q,q_{j+1},\dots,q_n)]\!]_k(s').$$

If $t = y$, then

$$\begin{aligned}
[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) &= s' : [\![p]\!]_k(s') && \text{def. of } [\![-]\!]\\
&= s' : [\![q]\!]_k(s') && \text{since } p \simeq_{k,s'} q\\
&= [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s) && \text{def. of } [\![-]\!].
\end{aligned}$$

Finally, suppose that the rule applying to s and f is non-receiving and thus of the form

$$\frac{l_1 \quad l_2 \quad \cdots \quad l_n}{s, \mathsf{f}(x_1,\dots,x_n) \to s', t} \qquad \text{where } t \in \Sigma^\star(\{x_1,\dots,x_n\} \setminus \{x_j\}) \text{ or } t = *.$$

If $t = *$, we have

$$[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) = s' = [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s).$$

Otherwise,

$$\begin{aligned}
[\![\mathsf{f}(p_1,\dots,p_n)]\!]_{k+1}(s) &= s' : [\![t(p_1,\dots,p_{j-1},p_{j+1},\dots,p_n)]\!]_k(s') && \text{def. of } [\![-]\!]\\
&= s' : [\![t(q_1,\dots,q_{j-1},q_{j+1},\dots,q_n)]\!]_k(s')\\
&= [\![\mathsf{f}(q_1,\dots,q_n)]\!]_{k+1}(s) && \text{def. of } [\![-]\!],
\end{aligned}$$

where in the second step we use that $p_m \simeq_k q_m$ for $m \neq j$ and that $\simeq_k$ is a congruence by induction. ◀

Proof of Corollary 5.5

Let $\mathcal{L}$ be a streamlined stateful SOS specification over the signature Σ.

1. Suppose first that the map $[-] = [-]_{\mathcal{L}} \colon \mu\Sigma \to \nu T$ is surjective. Let $\mathsf{f} \in \Sigma$ be an n-ary operator and suppose that $p_m, q_m \in \nu T$ $(m = 1, \ldots, n)$ satisfy $\mathsf{trc}(p_m) = \mathsf{trc}(q_m)$ for all m. Choose $\overline{p}_m, \overline{q}_m \in \mu\Sigma$ with $p_m = [\overline{p}_m]$ and $q_m = [\overline{q}_m]$. Then we have for every m that

   $$[\![\overline{p}_m]\!] = \mathsf{trc}([\overline{p}_m]) = \mathsf{trc}(p_m) = \mathsf{trc}(q_m) = \mathsf{trc}([\overline{q}_m]) = [\![\overline{q}_m]\!].$$

   Hence, $[\![\mathsf{f}(\overline{p}_1, \ldots, \overline{p}_m)]\!] = [\![\mathsf{f}(\overline{q}_1, \ldots, \overline{q}_m)]\!]$ since the kernel $\simeq$ of $[\![-]\!]$ is a congruence by Theorem 5.4. It follows that

   $$\mathsf{trc}(\mathsf{f}(p_1, \ldots, p_m)) = \mathsf{trc}(\mathsf{f}([\overline{p}_1], \ldots, [\overline{p}_n])) = \mathsf{trc}([\mathsf{f}(\overline{p}_1, \ldots, \overline{p}_n)]) = [\![\mathsf{f}(\overline{p}_1, \ldots, \overline{p}_n)]\!]$$

   where the second equality uses that $[\![-]\!]$ is a Σ-homomorphism. Similarly

   $$\mathsf{trc}(\mathsf{f}(q_1, \ldots, q_m)) = [\![\mathsf{f}(\overline{q}_1, \ldots, \overline{q}_n)]\!].$$

   Thus $\mathsf{trc}(\mathsf{f}(p_1, \ldots, p_m)) = \mathsf{trc}(\mathsf{f}(q_1, \ldots, q_m))$, proving that the kernel of trc is a congruence.
2. We now show that the general case can be reduced to the situation in item 1. To this end, we extend the given specification $\mathcal{L}$ to a specification $\mathcal{L}'$ as follows:
   - Extend Σ to a signature Σ' by adding a constant symbol c_t for every $t \in \nu T$.
   - Add the new rules

     $$\frac{}{s, c_t \to s', c_{t'}} \qquad \text{for every } s \in S \text{ and } t \in \nu T,$$

     where s' is the label of the edge from the root of t to its s-th child and t' is the corresponding subtree.

   Clearly the extended specification $\mathcal{L}'$ is streamlined since constants are passive operators, on which the streamlined format puts no restrictions. Moreover $[c_t]_{\mathcal{L}'} = t$ for every $t \in \nu T$, hence the map $[-]_{\mathcal{L}'}$ is surjective. By item 1 applied to $\mathcal{L}'$ we know that the kernel of trc is congruence w.r.t. the Σ'-algebra structure on νT. In particular, it is a congruence w.r.t. its Σ-algebra structure. ◀

Proof of Theorem 5.11

For every pair of complex terms $p, q \in \Sigma^\star(\mu\Sigma)$ we write $(p, q) \in C[\approx]$ if p and q have the same shape (i.e. they are equal after forgetting the labels of $\mu\Sigma$-labelled leaves) and matching leaves from $\mu\Sigma$ are pairwise $\approx$-equivalent. To prove that $\approx$ is a congruence, we need to show

$$(p, q) \in C[\approx] \implies \hat{\iota}(p) \approx \hat{\iota}(q) \qquad \text{for all } p, q \in \mu\Sigma,$$

where $\hat{\iota} \colon \Sigma^\star(\mu\Sigma) \to \mu\Sigma$ is the term evaluation map. By symmetry, it suffices to prove the following statement for all $s, \overline{s} \in S$:

$$\text{If } (p, q) \in C[\approx] \text{ and } s, \hat{\iota}(p) \text{ terminates in state } \overline{s}, \text{ then } s, \hat{\iota}(q) \text{ terminates in state } \overline{s}. \quad \text{(A.5)}$$

The proof proceeds by induction on the number k of steps until termination of $s, \hat{\iota}(p)$. Note that we count a terminating step $s, r \downarrow \overline{s}$ as one computation step.

For $k = 0$ the statement is vacuously true since no program terminates after 0 steps.

For the induction step, let $k > 0$ and suppose that $s, \hat{\iota}(p)$ terminates after k steps in the state $\overline{s}$. We need to show that also $s, \hat{\iota}(q)$ terminates in $\overline{s}$. This is shown by induction on the structure of p:

- If p is a leaf of the term tree, i.e. an element of $\mu\Sigma$, then $(p,q) \in C[\approx]$ implies that q has the same form and $p \approx q$. It follows that $\hat{\iota}(p) \approx \hat{\iota}(q)$, since $\hat{\iota}(p) = p$ and $\hat{\iota}(q) = q$.
- Now suppose that $p = \mathsf{f}(p_1, \dots, p_n)$ and $q = \mathsf{f}(q_1, \dots, q_n)$ for some n-ary operator f, $n > 0$. From $(p,q) \in C[\approx]$, it follows that $(p_i, q_i) \in C[\approx]$ for $i = 1, \dots, n$. We distinguish two cases:

1. The operator f is passive. Then s, p and s, q trigger the same rule

$$\frac{}{s, \mathsf{f}(x_1, \dots, x_n) \to s', t} \qquad \text{where } t \in \Sigma^\star(\{x_1, \dots, x_n\}) \text{ or } t = *.$$

   If the rule is terminating ($t = *$) then both $\hat{\iota}(p)$ and $\hat{\iota}(q)$ terminate after one step in state $\overline{s} = s'$. Otherwise, their respective first computation steps are

$$s, \hat{\iota}(p) \to s', \hat{\iota}(p') \qquad \text{and} \qquad s, \hat{\iota}(q) \to s', \hat{\iota}(q')$$

   for $p', q' \in \Sigma^\star(\mu\Sigma)$ given by $p' = t[p_1/x_1, \dots, p_n/x_n]$ and $q' = t[q_1/x_1, \dots, q_n/x_n]$. Since $(p_i, q_i) \in C[\approx]$ for $i = 1, \dots, n$, we have $(p', q') \in C[\approx]$. Moreover, $s', \hat{\iota}(p')$ terminates in the state $\overline{s}$ in strictly less than k steps, so by induction, $s', \hat{\iota}(q')$ terminates in the state $\overline{s}$. It follows that also $s, \hat{\iota}(q)$ terminates in the state $\overline{s}$.
2. The operator f is active with receiving position j. Then $s, \hat{\iota}(p)$ and $s, \hat{\iota}(q)$ will have to execute their subterms $\hat{\iota}(p_j)$ and $\hat{\iota}(q_j)$, respectively. Executing $s, \hat{\iota}(p_j)$ requires at most k steps and terminates in some state s'. Since $(p_j, q_j) \in C[\approx]$, the inner induction hypothesis implies that $s, \hat{\iota}(q_j)$ also terminates in s'. Upon termination of $\hat{\iota}(p_j)$ or $\hat{\iota}(q_j)$, respectively, both $s, \hat{\iota}(p)$ and $s, \hat{\iota}(q)$ therefore trigger a rule

$$\frac{?, x_j \downarrow s'}{?, \mathsf{f}(x_1, \dots, x_n) \to s'', t} \qquad \text{where } t \in \Sigma^\star(\{x_1, \dots, x_n\} \setminus \{x_j\}) \text{ or } t = *.$$

   Note that while the input ? might be different for the two computations, the output s'' and target t, respectively, are the same for both computations because they only depend on s'.
   If the above rule is terminating ($t = *$), then $s, \hat{\iota}(p)$ and $s, \hat{\iota}(q)$ both terminate in the state $\overline{s} = s''$, and we are done. Otherwise, $s, \hat{\iota}(p)$ evolves to $s'', \hat{\iota}(p')$, and $s, \hat{\iota}(q)$ evolves to $s'', \hat{\iota}(q')$ in finitely many steps, where $p', q' \in \Sigma^\star(\mu\Sigma)$ are given by $p' = t[p_m/x_m : m \neq j]$ and $q' = t[q_m/x_m : m \neq j]$; moreover, $s'', \hat{\iota}(p')$ terminates in state $\overline{s}$ in strictly less than k steps. Since $(p_i, q_i) \in C[\approx]$ for all i, we have $(p', q') \in C[\approx]$. By the outer induction hypothesis, we conclude that $s'', \hat{\iota}(q')$ terminates in $\overline{s}$. Consequently, also $s, \hat{\iota}(q)$ terminates in $\overline{s}$. ◀

Proof of Corollary 5.12

The proof is completely analogous to the one of Corollary 5.5; just note that extending a cool stateful SOS specification by constants preserves coolness. ◀