Friedrich-Alexander-Universität
Faculty of Engineering

Bisimulation-Based Process Algebra in Higher-Order GSOS

Master's Thesis

Florian Guthmann
florian.guthmann@fau.de

Advisor: PD Dr. Sergey Goncharov

13th February 2025

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 13. Februar 2025

Florian Guthmann

Contents

Abstract

Process algebras are mathematical frameworks for modeling the behaviour of concurrent systems. Bisimilarity serves as an equivalence relation for comparing their behaviour. GSOS is a rule format for defining the operational semantics for such process algebras, but is limited in its ability to encode process algebras with recursion. The key semantic property of bisimilarity being a congruence is hard to establish in the presence of recursion – the denotational methods employed for that are notoriously involved and fragile. Higher-Order GSOS is a recently proposed framework helping to resolve these limitations. This master's thesis proposes a treatment of process algebras with recursion via (properly) higher-order abstract GSOS, using Milner's Calculus of Communicating Systems as a concrete example. This is done by specifying alternative operational semantics rules, showing their equivalence to the standard semantics in the guarded case, and giving the corresponding categorical interpretation to fit into higher-order abstract GSOS.

1 Introduction

In the field of process algebra, one of the main ways to define the concept of equivalence between processes is (strong) *bisimulation*. Other notions exist, but bisimilarity serves as the finest equivalence relation between processes. It is desirable for bisimilarity to be a *congruence*: If two terms are bisimilar, they cannot be distinguished by any context. Proofs of congruence in the presence of general recursion are notoriously involved, however. One possible way to deal with this problem is by a detour through denotational semantics. This requires establishing a connection between operational and denotational semantics, which is hard even for simple languages. The denotational model also has to be carefully crafted for every case and may have to be redesigned even for small changes like switching from finite to infinite non-determinism [AP86]. For process algebras, the denotational domain of labelled transition systems necessarily involves a degree of non-determinism, so domain-theoretic proofs are made more complicated and brittle [AC98; JT98].

An alternative approach is to use purely operational methods to tackle the issue. Proving compositionality for such higher-order languages in the operational semantics can be a problem too. The abstract GSOS framework presented by Turi and Plotkin [TP97] gives compositionality proofs for languages covered by its rule format. This includes process algebras without fixpoints, where compositionality ensures that bisimulation is in fact a congruence. A limitation of the rule format is that arbitrary recursion cannot be encoded. The recently developed higher-order extension of GSOS, presented by Goncharov et al. [Gon+24] provides additional expressivity to cater to higher-order languages, although applications to process algebra with recursion has not been explored so far. In this case, non-determinism complicates the theory, since bisimulation does *not* reduce to mutual simulation.

One such a process algebra based on bisimulation is the Calculus of Communicating Systems (CCS). We will focus on this system, although the methodology could be used for other bisimulation-based process algebras. We solve the problem of encoding recursion by reformulating the original specification of CCS in such a way that it fits the rule format of higher-order GSOS. This allows us to obtain a compositional operational model where bisimulation is a congruence. In Chapter 2, we will begin by introducing the necessary background concerning CCS and abstract GSOS, both first-order and higher-order. Following that, in Chapter 3 we define an alternative operational semantics for CCS and prove its equivalence to the standard semantics. This is then used to define the semantics in the framework of higher-order abstract GSOS and obtain the compositional operational model we are seeking.

Large parts of this thesis are formalized in the proof assistant Agda [Agd]. Theorems and definitions that are part of the formalization are marked with the Agda logo ( ) and their respective identifiers. A more in-depth discussion of the implementation can be found in Chapter 4.

2 Background

This section will provide the necessary background for this thesis. We will cover the basics of process algebra using the example of the Calculus of Communicating Systems. Afterwards, the framework of first-order abstract GSOS will be introduced along with its extension of higher-order abstract GSOS. But first, let us review the required background in category theory.

2.1 Categorical Preliminaries

We assume the reader is familiar with basic notions of category theory such as functors, natural transformations and monads. This section reviews some terminology and notations used in the thesis.

Products and Coproducts Given two objects A, B in a category $\mathcal{C}$, we write $A \times B$ for a selected product, i.e. an object satisfying the universal property

$$\begin{tikzcd} & C \arrow[dl, "g"'] \arrow[d, dashed, "\exists! h"] \arrow[dr, "f"] & \\ A & A \times B \arrow[l, "\mathrm{fst}"] \arrow[r, "\mathrm{snd}"'] & B \end{tikzcd} \tag{2.1}$$

where $\mathrm{fst}\colon A \times B \to A$ and $\mathrm{snd}\colon A \times B \to B$ are the projections. Since h is uniquely determined by f and g, we will write $\langle f, g\rangle$ instead.

Dually, we write $A + B$ for a selected coproduct, i.e. an object satisfying the universal property

$$\begin{tikzcd} & C & \\ A \arrow[ur, "f"] \arrow[r, "\mathrm{inl}"'] & A + B \arrow[u, dashed, "\exists! h"] & B \arrow[ul, "g"'] \arrow[l, "\mathrm{inr}"] \end{tikzcd} \tag{2.2}$$

where $\mathrm{inl}\colon A \to A + B$ and $\mathrm{inr}\colon B \to A + B$ are the injections. Since h is uniquely determined by f and g, we write $[f, g]$ instead. Furthermore, we let $\nabla_X : X + X \to X$ denote the codiagonal defined by $\nabla_X := [\mathrm{id}_X, \mathrm{id}_X]$.

Algebras over a Functor Given an endofunctor $F\colon \mathcal{C} \to \mathcal{C}$ on a category $\mathcal{C}$, an *algebra over* F (or F-algebra) is a pair (A, a) where A is an object of $\mathcal{C}$ (the *carrier* of the algebra) and a is a morphism $a\colon FA \to A$ (its *structure*). A morphism $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$ is an F*-algebra homomorphism* between F-algebras (A, a) and (B, b) if the following diagram commutes:

$$\begin{tikzcd} FA \arrow[r, "a"] \arrow[d, "Ff"] & A \arrow[d, "f"] \\ FB \arrow[r, "b"] & B \end{tikzcd} \tag{2.3}$$

When the structure is clear, we often ignore the notational difference between A and (A, a) and write an F-algebra homomorphism $f\colon (A, a) \to (B, b)$ as $f\colon A \to B$. F-algebras and F-algebra homomorphisms form a category $\mathsf{alg}(F)$. We denote the initial object in that category, if it exists, by $(\mu F, \iota)$. The unique F-algebra homomorphism from the initial F-algebra to any F-algebra (A, a) is denoted as

$$\textbf{it}\ a \colon (\mu F, \iota) \to (A, a) \tag{2.4}$$

Definition 2.1. A *free F-algebra* on an object $X \in \mathrm{Ob}(\mathcal{C})$ is an F-algebra $(F^\star X, \iota_X)$ such that for any F-algebra (A, a) and morphism $h \in \mathrm{Hom}_{\mathcal{C}}(X, A)$ there exists a unique F-algebra homomorphism $h^\star \colon F^\star X \to A$ with the universal property:

$$\begin{tikzcd} FF^\star X \arrow[r, "\iota_X"] \arrow[dd, "Fh^\star"] & F^\star X \arrow[dd, dashed, "\exists!\, h^\star"] & \\ & & X \arrow[ul, "\eta_X"'] \arrow[dl, "h"] \\ FA \arrow[r, "a"] & A & \end{tikzcd} \tag{2.5}$$

Definition 2.2. If every object $X \in \mathrm{Ob}(\mathcal{C})$ generates a free F-algebra, then F generates a *free monad* $F^\star \colon \mathcal{C} \to \mathcal{C}$ [Bar70], that assigns to

- an object $X \in \mathrm{Ob}(C)$ the carrier of its free F-algebra $F^\star X$
- a morphism $f \colon X \to Y$ the morphism $(\eta_Y \circ f)^\star$ given by the universal property of the free F-algebra on X.

Definition 2.3. An *Eilenberg-Moore algebra* over a monad (T, η, μ) is a morphism $\hat{a} \colon TX \to X$ such that

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\hat{a}"] \\ & X \end{tikzcd} \qquad \begin{tikzcd} TTX \arrow[r, "\mu_X"] \arrow[d, "T\hat{a}"] & TX \arrow[d, "\hat{a}"] \\ TX \arrow[r, "\hat{a}"] & X \end{tikzcd} \tag{2.6}$$

For a functor F that generates a free monad $F^\star$, every F-algebra (A, a) induces an Eilenberg-Moore algebra $\hat{a} \colon F^\star A \to A$, given by the universal property of the free algebra on A:

$$\begin{tikzcd} FF^\star A \arrow[r, "\iota_A"] \arrow[dd, "F\hat{a}"'] & F^\star A \arrow[dd, dashed, "\hat{a} := \mathrm{id}_A^\star"'] & \\ & & A \arrow[ul, "\eta_A"'] \arrow[dl, "\mathrm{id}_A"] \\ FA \arrow[r, "a"] & A & \end{tikzcd} \tag{2.7}$$

Signature Functors An algebraic signature consist of a set $\overline{\Sigma}$ of symbols and a map $\mathrm{ar} \colon \overline{\Sigma} \to \mathbb{N}$, that assigns to every symbol $f \in \overline{\Sigma}$ its *arity* $\mathrm{ar}(f)$. Symbols of arity 0 are called *constants*, those of arity 1 are called *unary function symbols* and those of arity 2 are called *binary function symbols*.

Every signature $\overline{\Sigma}$ induces a polynomial endofunctor

$$\Sigma X = \coprod_{f \in \overline{\Sigma}} X^{\mathrm{ar}(f)} \tag{2.8}$$

A Σ-algebra is then an object A together with a morphism $f_A \colon A^n \to A$ for every n-ary function symbol in $\overline{\Sigma}$. For a set X of variables, the free algebra $\Sigma^\star X$ consists of terms over the signature $\overline{\Sigma}$ with variables from X. The free algebra on the empty set (or more generally, the initial object) is the set of *closed* terms, i.e terms with no free variable occurrences.

Definition 2.4. A relation $\sim \subseteq A \times A$ on a Σ-algebra (A, a) is called a Σ-*congruence* if for every $f \in \overline{\Sigma}$ with $\mathrm{ar}(f) = n$ and $a_i, b_i \in A, 1 \leq i \leq n$,

$$a_i \sim b_i \text{ (for all } 1 \leq i \leq n) \implies f_A(a_1, \ldots, a_n) \sim f_A(b_1, \ldots, b_n) \tag{2.9}$$

Intuitively, a Σ-congruence $\sim$ is a relation on terms such that if two terms are related, they cannot be distinguished by any context.

Coalgebras over a Functor Given an endofunctor $F\colon \mathcal{C} \to \mathcal{C}$ on a category $\mathcal{C}$, a *coalgebra over* F (or F-coalgebra) is a pair (C, c) consisting of an object C (the *state space* of the algebra) and a morphism $c\colon C \to FC$ (its *structure*). A morphism $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$ is an F-*coalgebra homomorphism* between F-coalgebras (A, a) and (B, b) if the following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "a"] \arrow[d, "f"] & FA \arrow[d, "Ff"] \\
B \arrow[r, "b"] & FB
\end{tikzcd} \tag{2.10}$$

As with F-algebras, we will often omit the structure of coalgebras and write an F-coalgebra homomorphism $f\colon (A, a) \to (B, b)$ as $f\colon A \to B$. F-coalgebras and their homomorphisms form a category $\mathsf{coalg}(F)$. We denote the terminal (or final) object in that category, if it exists, by $(\nu F, \tau)$. The unique F-coalgebra homomorphism from any F-coalgebra (C, c) to the terminal coalgebra is denoted as

$$\mathbf{coit}\ c\colon (C, c) \to (\nu F, \tau) \tag{2.11}$$

We will use coalgebras to model state-based systems categorically. In particular, coalgebras of the (covariant) powerset functor $\mathcal{P}\colon \mathsf{Set} \to \mathsf{Set}$ can be used to model nondeterministic systems. Informally, the final coalgebra νF consist of the abstract behaviours of all F-coalgebras.

Definition 2.5. A relation $R \subseteq C \times D$ on F-coalgebras (C, c) and (D, d) is a *bisimulation* if there exists a coalgebraic structure $r\colon R \to FR$ such that

$$\begin{tikzcd}
C \arrow[d, "c"] & R \arrow[l, "\mathrm{fst}|_R"'] \arrow[r, "\mathrm{snd}|_R"] \arrow[d, "r"] & D \arrow[d, "d"] \\
FC & FR \arrow[l, "F(\mathrm{fst}|_R)"] \arrow[r, "F(\mathrm{snd}|_R)"'] & FD
\end{tikzcd} \tag{2.12}$$

The fact that the diagram above commutes corresponds to the projections $\mathrm{fst}|_R$ and $\mathrm{snd}|_R$ being F-coalgebra homomorphisms.

Dinatural Transformations Natural transformations are the primary tool one can use to ensure on an abstract level, that a map is parametrically polymorphic, i.e. that it does not inspect the structure of its arguments. In a natural transformation $\alpha\colon F \Rightarrow G$, both functors F and G depend on some X with the same variance (either covariantly or contravariantly). If we want F and G to be mixed-variance bifunctors, this becomes limiting: While we can state natural transformations between F and G when either one of their arguments is fixed, we cannot abstract over both arguments. If we restrict to families of maps where F and G depend on some X that is used both covariantly *and* contravariantly, we arrive at a useful abstraction.

Definition 2.6. Let $F, G \colon \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathcal{C}$ be bifunctors of mixed variance. A *dinatural transformation* $\alpha \colon F \overset{\bullet}{\Rightarrow} G$ is a family of morphisms $\alpha_X \colon F(X,X) \to G(X,X)$ such that

$$\begin{tikzcd}
& F(X,X) \arrow[r, "\alpha_X"] & G(X,X) \arrow[dr, "G(\mathrm{id}_X, f)"] & \\
F(Y,X) \arrow[ur, "F(f, \mathrm{id}_X)"] \arrow[dr, "F(\mathrm{id}_Y, f)"'] & & & G(X,Y) \\
& F(Y,Y) \arrow[r, "\alpha_Y"] & G(Y,Y) \arrow[ur, "G(f, \mathrm{id}_Y)"'] &
\end{tikzcd} \tag{2.13}$$

commutes for every $f \colon X \to Y$.

This concludes our review of the necessary categorical background.

2.2 Process Algebra

The field of process algebra studies the behaviour of concurrent processes in a mathematical model. A “process” in this context is a system of which a behaviour may be observed, such as the execution of software in a computer or the actions of a machine. We think of these behaviours as being comprised of discrete “actions” which are separated in time. A simple model of a process is a mere map between inputs and outputs. In this case, automata theory serves as an algebraic model that allows for equational reasoning over automata. In process algebra however, the goal is to model *concurrent* systems, i.e. processes that interact with one another during execution and run in parallel. This usually comes in the form of a “parallel composition” operator that is included in the language of expressions. Examples of such process algebras include the aforementioned “Calculus of Communicating Systems” by Milner [Mil80], the theory of “Communicating sequential processes” (CSP) by Hoare [Hoa78], the “Algebra of communicating processes” (ACP) by Bergstra and Klop [BK82] and the π-calculus by Milner [Mil99].

We will concern ourselves only with CCS, but note that the general approach could be adapted to any process algebra with similar features.

2.2.1 CCS

The Calculus of Communicating Systems is a process algebra introduced by Milner [Mil80; Mil89]. Its basic construct are *processes* that can emit *actions*, stepping to a new process. For instance, we will write $P \xrightarrow{\alpha} Q$ to denote that a process P takes a step, emitting the action α and afterwards behaves like the process Q. To describe how such processes may behave and interact, CCS features a number of concepts:

Deadlock We will denote the deadlocked process that cannot take any steps by $\varnothing$.

Action Prefixing A process that emits an action α and afterwards behaves like some process P is denoted by prefixing the action: $\alpha \bullet P$.

Non-deterministic Choice Given two processes P and Q, the process $P + Q$ chooses a possible step in either P or Q to some process R and behaves like R afterwards.

Parallel Composition Given two processes P and Q, we can interleave them in the process $P|Q$. This process allows either P or Q to take a step. Unlike the non-deterministic choice operator, the resulting process is still a composition of the result of one process taking a step and the other process, which is left untouched. If both processes take “opposing” steps, the composed process takes a synchronisation step and emits the designated silent action τ.

Restriction We can restrict what actions a process P is allowed to emit: The process $P \setminus L$ can only take steps with actions which are not in the set L. This is mainly used to force P into taking silent transitions.

Renaming If we want to rename the actions a process P via some function φ, the renaming construct $P\,[\varphi]$ does the right thing: If $P \xrightarrow{\alpha} Q$ then $P\,[\varphi] \xrightarrow{\varphi(\alpha)} Q\,[\varphi]$.

Recursion Finally, CCS allows processes to be defined recursively. Unlike Milner, who used recursive process equations to accomplish this, we will equivalently use a fixpoint operator, akin to an abstraction in the λ-calculus. For instance, the process defined by the recursive equation $P = \alpha \bullet P$ can be represented simply by $(\mathrm{fix}\, X.\, (\alpha \bullet X))$.

Actions Before we examine the syntax of CCS, we first need to define a set of actions over which the language is parametric. It is required that there exists an involution (called the complement) on actions. Therefore, we fix a set $\mathcal{A}$ of actions with

- $\tau \in \mathcal{A}$ representing the silent action
- a map $\overline{\cdot} \colon \mathcal{A} \setminus \{\tau\} \to \mathcal{A} \setminus \{\tau\}$ that is an involution (i.e. $\overline{\overline{\alpha}} = \alpha$ for any $\alpha \in \mathcal{A}$). We refer to $\overline{\alpha}$ as the *complement* of α.

A common interpretation is that an action α corresponds to an “input” of type α while its complement $\overline{\alpha}$ corresponds to an “output” of type α. Its main use, however, is to facilitate the synchronisation between processes, where a “handshake” occurs when one process emits an action while another emits its complement. In that case, the process composed of both subprocesses emits a silent action.

CCS also features the ability to rename the actions a process emits. This is done via a function φ on the set of actions. However, we do require that φ interacts nicely with the complement on actions.

Definition 2.7. A *renaming function* $\varphi \colon \mathcal{A} \setminus \{\tau\} \to \mathcal{A} \setminus \{\tau\}$ is a function on actions that respects complementation:

$$\forall \alpha \in \mathcal{A}.\ \varphi(\overline{\alpha}) = \overline{\varphi(\alpha)} \tag{2.14}$$

We call $\mathsf{Ren}(\mathcal{A})$ the set of renaming functions on $\mathcal{A}$.

Syntax Finally, we can state the syntax of CCS, which is parametric over a set $\mathcal{V}$ of variables:

$$P, Q \coloneqq \varnothing \mid X \mid \alpha \bullet P \mid P + Q \mid P|Q \mid P \setminus L \mid P\,[\varphi] \mid \mathrm{fix}\, X.\, P \tag{2.15}$$

where $X \in \mathcal{V}$, $\alpha \in \mathcal{A}$, $L \subseteq \mathcal{A} \setminus \{\tau\}$, $\varphi \in \mathsf{Ren}(\mathcal{A})$.

To avoid excessive use of parentheses, we assume the operator precedence

$$\begin{Bmatrix} (-)\,[\varphi] \\ (-) \setminus L \end{Bmatrix} \quad > \quad \alpha \bullet (-) \quad > \quad (-)|(-) \quad > \quad (-) + (-) \quad > \quad \mathrm{fix}\, X.\, (-) \tag{2.16}$$

De Bruijn-Indices Instead of using this syntax with concrete variable names, we will instead employ a commonly used formalization trick invented by de Bruijn [de 72], originally used for the λ-calculus. This consists of fixing the set of variables to be the natural numbers. Furthermore, we require that a fixpoint operator *always* binds the variable $0 \in \mathbb{N}$. In order for this to work, occurrences of variables are converted to *indices*: An index $\#m$ denotes that one has to syntactically "step over" m fixpoint operators to find the corresponding binder. If it exceeds all surrounding binders, then $\#m$ is a free variable.

To utilize de Bruijn-indices, we need to slightly change the syntax:

$$P, Q \coloneqq \varnothing \mid \#n \mid \alpha \bullet P \mid P + Q \mid P|Q \mid P \setminus L \mid P\,[\varphi] \mid \text{fix } P \tag{2.17}$$

where $n \in \mathbb{N}$, $\alpha \in \mathcal{A}$, $L \subseteq \mathcal{A} \setminus \{\tau\}$, $\varphi \in \mathsf{Ren}(\mathcal{A})$. Notice how the fixpoint operator no longer accepts an argument for the bound variable, since it is always $\#0$. The translation to and from the concrete syntax in Equation (2.15) is straightforward: For instance, consider this example term and its translation into the syntax with de Bruijn-indices:

$$\text{fix } X.\, \text{fix } Y.\, (\alpha \bullet X) | Y \quad \equiv \quad \text{fix fix } (\alpha \bullet \#1) | \#0$$

From this point onward, we will use the syntax with de Bruijn-indices exclusively. The set of terms over the grammar in Equation (2.17) is denoted as Proc.

A substitution is a function of type $m \to \text{Proc}$ for some $m \in \mathbb{N}$. We fix the set $\mathsf{Subst} \coloneqq \coprod_{n \in \mathbb{N}} \text{Proc}^n$ to include all substitutions. A substitution $\sigma \in \mathsf{Subst}$ can be "lifted" to a substitution $\uparrow \sigma$ by incrementing all indices and inserting the variable $\#0$ at 0:

$$\begin{aligned} (\uparrow \sigma)(0) &= \#0 \\ (\uparrow \sigma)(n+1) &= \sigma(n) \end{aligned} \tag{2.18}$$

The most common instance of a substitution we will encounter is one that substitutes the 0th variable with some term Q:

$$\begin{aligned} ([0 \mapsto Q])(0) &= Q \\ ([0 \mapsto Q])(n+1) &= n \end{aligned} \tag{2.19}$$

Applying a substitution σ to a term is illustrated in Figure 2.1:

$$\frac{P' = P\,\sigma}{(\alpha \bullet P)\sigma = \alpha \bullet P'}\ \text{ACT}_{\text{SUB}} \qquad \frac{}{X\sigma = \sigma(X)}\ \text{NAME}_{\text{SUB}}$$

$$\frac{P\sigma = P' \quad Q\sigma = Q'}{(P + Q)\sigma = P' + Q'}\ \text{SUM}_{\text{SUB}} \qquad \frac{P\sigma = P' \quad Q\sigma = Q'}{(P|Q)\sigma = P'|Q'}\ \text{PAR}_{\text{SUB}}$$

$$\frac{P\sigma = P'}{(P \setminus L)\sigma = P' \setminus L}\ \text{RES}_{\text{SUB}} \qquad \frac{P\sigma = P'}{(P\,[\varphi])\sigma = P'\,[\varphi]}\ \text{REN}_{\text{SUB}}$$

$$\frac{P(\uparrow \sigma) = P'}{(\text{fix } P)\,\sigma = \text{fix } P'}\ \text{FIX}_{\text{SUB}}$$

Figure 2.1: Semantics of substitution application

We will need some meta theory about substitutions now:

Theorem 2.8 (`subst-commute`). *For a substitution σ and terms P and Q the following identity holds:*

$$(P\,[0 \mapsto (Q\,\sigma)])\,(\uparrow \sigma) = (P\,[0 \mapsto Q])\,\sigma \tag{2.20}$$

Proof. This is proven by building an algebra of substitutions and reducing the problem to identities in that algebra. The proof can be found in the Agda formalization or, in a slightly different form, in the Substitution chapter of Programming Language Foundations in Agda [WKS22]. □

Operational Semantics Now that we have defined how to syntactically build terms of CCS, it remains to describe how these processes *behave*. This is done by defining a set of rules, referred together as an operational semantics. These rules formally define our informal concepts of how the syntactical constructs of CCS should behave. Let us highlight some of the rules shown in Figure 2.2 that are of interest:

The ACT **rule:** As expected, a process prefixed with an action α can take an α-step without any premises.

The SYNC **rule:** Here we see how processes can interact: A handshake is performed when P takes an α-step to P' while Q takes an $\overline{\alpha}$-step to Q'. The parallel composition of P and Q then takes a τ-step to $P'|Q'$.

The FIX **rule:** The most interesting rule deals with the semantics of the fixpoint operator: For a process fix P to take a step, we need to prove that the process resulting from substituting the bound variable $\#0$ with the original term fix P takes the same step. This makes intuitive sense by considering how fixpoint operators correspond to recursive equations. The FIX rule then simply unfolds one layer of the equation.

$$\dfrac{}{\alpha\bullet P \xrightarrow{\alpha} P}\ \text{ACT}$$

$$\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}\ \text{SUM}_\text{l} \qquad \dfrac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}\ \text{SUM}_\text{r}$$

$$\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ \text{PAR}_\text{L} \qquad \dfrac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}\ \text{PAR}_\text{R}$$

$$\dfrac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}\ \text{SYNC}$$

$$\dfrac{P \xrightarrow{\alpha} P' \qquad \alpha, \overline{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha} P'}\ \text{RES} \qquad \dfrac{P \xrightarrow{\alpha} P'}{P\,[\varphi] \xrightarrow{\varphi(\alpha)} P'\,[\varphi]}\ \text{REN}$$

$$\dfrac{P\,[0 \mapsto \text{fix } P] \xrightarrow{\alpha} P'}{\text{fix } P \xrightarrow{\alpha} P'}\ \text{FIX}$$

Figure 2.2: Operational Semantics of CCS (`Step.Standard`)

We can use this operational semantics to build derivation trees, such as the one below, which are used as proofs that a proposed step is valid for some process.

Example 2.9. Consider the CCS term $P \coloneqq \text{fix } (\text{fix } \alpha\bullet\#1)$. We can show that P takes an α-step to itself:

$$\dfrac{\dfrac{\dfrac{}{((\alpha\bullet(\text{fix } (\text{fix } \alpha\bullet\#1)))\,[0 \mapsto (\text{fix } \alpha\bullet\#1)]) = (\alpha\bullet(\text{fix } (\text{fix } \alpha\bullet\#1))) \xrightarrow{\alpha} \text{fix } (\text{fix } \alpha\bullet\#1)}\ \text{ACT}}{((\text{fix } \alpha\bullet\#1)\,[0 \mapsto \text{fix } (\text{fix } \alpha\bullet\#1)]) = (\text{fix } \alpha\bullet(\text{fix } (\text{fix } \alpha\bullet\#1))) \xrightarrow{\alpha} \text{fix } (\text{fix } \alpha\bullet\#1)}\ \text{FIX}}{\text{fix } (\text{fix } \alpha\bullet\#1) \xrightarrow{\alpha} \text{fix } (\text{fix } \alpha\bullet\#1)}\ \text{FIX}$$

Definition 2.10. The rules in Figure 2.2 describe how a process correspond to a *labelled transition system* (LTS). That is a tuple $\left(\mathrm{Proc}, \mathcal{A}, \left\{ \xrightarrow{\alpha} \,\middle|\, \alpha \in \mathcal{A} \right\}\right)$ with a designated state $P \in \mathrm{Proc}$. We only consider the part of the LTS that is reachable via $\longrightarrow$ from P. An LTS is called *finitely branching* if the set $\left\{ Q' \in \mathrm{Proc} \,\middle|\, Q \xrightarrow{\alpha} Q', \alpha \in \mathcal{A} \right\}$ is finite for all $Q \in \mathrm{Proc}$.

2.2.2 Equivalence of Processes

Now that we can describe the behaviour of processes with operational semantics, the question of how to state that two processes behave "equivalently" naturally arises. The issue is with finding an equivalence relation which satisfies our intuitive understanding.

Trace equivalence A simple approach is to relate processes for which the set of action sequences found in their behaviour is equal. We define a *trace* of a process P_0 to be a sequence of actions $(\alpha_k)_{0 \leq k}$ such that the transitions

$$P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} P_k$$

exists. Two processes P and Q are *trace-equivalent* if the set of traces from P and from Q are equal. This is an equivalence relation on processes, but it does not capture the intuitive understanding of equivalent processes, as the following example demonstrates.

Example 2.11. Consider these trace-equivalent processes:

$$P := (R|S) \setminus \{\alpha, \beta, \gamma\}$$
$$Q := (R|S') \setminus \{\alpha, \beta, \gamma\}$$

where

$$R := \text{fix } (\overline{\alpha} \bullet \beta \bullet \#0)$$
$$S := \text{fix } (\alpha \bullet ((\overline{\beta} \bullet \#0) + (\overline{\gamma} \bullet \#0)))$$
$$S' := \text{fix } ((\alpha \bullet \overline{\beta} \bullet \#0) + (\alpha \bullet \overline{\gamma} \bullet \#0))$$

Even though P and Q are trace-equivalent, they do seem to behave differently: In both processes, we can enforce a SYNC step via the restriction operator. In Q, this step already forces a choice for the left summand. In P this choice is left for the next step. This means that trace equivalence includes processes which intuitively behave differently.

Since trace equivalence is too coarse, we need to define a finer (or as it turns out the finest) equivalence relation on processes:

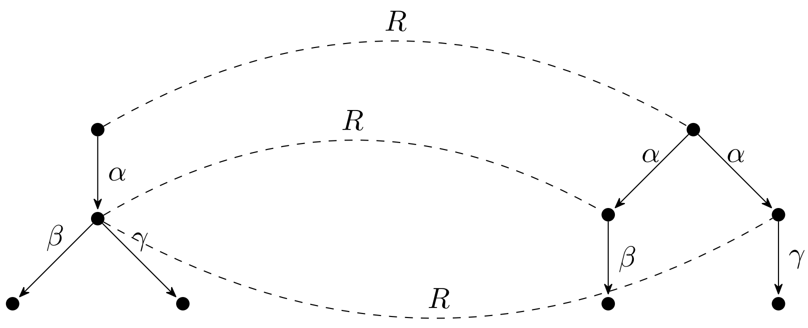
Definition 2.12. A relation $R \subseteq \mathrm{Proc} \times \mathrm{Proc}$ is a *strong bisimulation* if for any $(P, Q) \in R$ and P' such that $P \xrightarrow{\alpha} P'$ there exists a Q' with $Q \xrightarrow{\alpha} Q'$ and the same holds for the inverse relation R^-. Diagrammatically this can be expressed via a Forth and Back condition:

$$\begin{tikzcd} P \arrow[r, dash, "R"] \arrow[d, "\alpha"] & Q \arrow[d, dashed, "\alpha"] \\ P' \arrow[r, dash, "R"] & Q' \end{tikzcd} \qquad \begin{tikzcd} P \arrow[r, dash, "R"] \arrow[d, dashed, "\alpha"] & Q \arrow[d, "\alpha"] \\ P' \arrow[r, dash, "R"] & Q' \end{tikzcd} \tag{2.21}$$

Forth Back

Two processes P and Q are *bisimilar*, if there exists a bisimulation R with $(P, Q) \in R$, which we denote by $P \sim Q$.

Example 2.13. Consider the processes $P \coloneqq \alpha \bullet (\beta \bullet \varnothing + \gamma \bullet \varnothing)$ and $Q \coloneqq (\alpha \bullet \beta \bullet \varnothing) + (\alpha \bullet \beta \bullet \varnothing)$. These produce the following LTS's:



It is easy to see that these LTSs are trace-equivalent. However, they are not bisimilar since constructing a bisimulation R fails in the bottom two states in R, which do not satisfy the Back and Forth conditions.

2.2.3 Guarded Processes

The rule for the fixpoint operator allows for a very general form of recursion. For instance, we can state processes with infinitely many outgoing transitions:

Example 2.14. Let $P \coloneqq \text{fix}\ ((\alpha \bullet \varnothing) | \#0)$. Then there are infinitely many possible α-transitions:

$$\begin{tikzcd}
P \arrow[d, "\alpha"'] \arrow[dr, "\alpha"] \arrow[drrr, "\alpha"] & & & & \\
(\varnothing | P) & (\alpha \bullet \varnothing | \varnothing | P) & \cdots & (\alpha \bullet \varnothing | \alpha \bullet \varnothing | \alpha \bullet \varnothing | \varnothing | P) & \cdots
\end{tikzcd}$$

These so-called "unguarded" processes [Mil83] pose problems in our formalization, such as making the transition relation $\xrightarrow{\alpha}$ undecidable [BIM95]. It is therefore sensible to work with a subset of processes where recursion is restricted to be "productive", i.e. recursive occurrences of variables are only allowed under an action prefix.

Definition 2.15 (`guarded`). A term $P \in \text{Proc}$ is *guarded*, if every occurrence of a variable is in a subterm of an action prefix. A concrete inductive definition can be found in Figure 2.3. Only in the rule for action prefixing, the guardedness of does P not simply follow from the guardedness of all subterms of P. Instead, $P \coloneqq \alpha \bullet Q$ is guarded, irregardless of the guardedness of Q.

Now we can state a fact about how guardedness interacts with substitution of variables:

Lemma 2.16 (`guarded-subst`). *Guardedness is preserved under substitution, i.e. for any guarded term P and substitution σ, the term $(P\,\sigma)$ is guarded.*

Proof. This follows directly by induction over the derivation of $\mathsf{guarded}(P)$. □

Let us finally prove that, when we disallow unguarded processes, we ensure that all processes have only finitely many outgoing transitions:

$$\frac{}{\mathsf{guarded}(\alpha \bullet P)} \qquad \frac{}{\mathsf{guarded}(\varnothing)} \qquad \frac{\mathsf{guarded}(P) \quad \mathsf{guarded}(Q)}{\mathsf{guarded}(P + Q)}$$

$$\frac{\mathsf{guarded}(P) \quad \mathsf{guarded}(Q)}{\mathsf{guarded}(P|Q)} \qquad \frac{\mathsf{guarded}(P)}{\mathsf{guarded}(P \setminus L)} \qquad \frac{\mathsf{guarded}(P)}{\mathsf{guarded}(P\,[\varphi])}$$

$$\frac{\mathsf{guarded}(P)}{\mathsf{guarded}(\text{fix } P)}$$

Figure 2.3: Guarded terms (`guarded`)

Proposition 2.17. *Any guarded term P produces a finitely branching LTS.*

Proof. By induction over P:

If $P = \varnothing$**, then** the set $\left\{P' \,\middle|\, \varnothing \xrightarrow{\alpha} P', \alpha \in \mathcal{A}\right\}$ is empty and therefore finite.

If $P = \#m$**, then** P is not guarded, contradicting the assumption.

If $P = \alpha \bullet Q$**, then** the set $\left\{P' \,\middle|\, \alpha \bullet Q \xrightarrow{\alpha} P', \alpha \in \mathcal{A}\right\} = \{Q\}$ is finite.

If $P = Q + R$**, then** the set

$$\begin{aligned} &\left\{P' \,\middle|\, Q + R \xrightarrow{\alpha} P', \alpha \in \mathcal{A}\right\} \\ = &\left\{Q' \,\middle|\, Q \xrightarrow{\alpha} Q', \alpha \in \mathcal{A}\right\} \cup \left\{R' \,\middle|\, R \xrightarrow{\alpha} R', \alpha \in \mathcal{A}\right\} \end{aligned}$$

is finite by applying the induction hypothesis to both Q and R.

The cases for sum, restriction and renaming follow analogously.

If $P = \text{fix } Q$**, then** the set

$$\begin{aligned} &\left\{P' \,\middle|\, \text{fix } Q \xrightarrow{\alpha} P', \alpha \in \mathcal{A}\right\} \\ = &\left\{P' \,\middle|\, (Q\,[0 \mapsto \text{fix } Q]) \xrightarrow{\alpha} P', \alpha \in \mathcal{A}\right\} \end{aligned}$$

is finite by applying the induction hypothesis to $Q\,[0 \mapsto \text{fix } Q]$, where guardedness is guaranteed by Lemma 2.16. □

2.3 Abstract GSOS

This section introduces both the first-order and the higher-order abstract GSOS frameworks. Both are categorical theories inspired by the GSOS rule format presented by Bloom, Istrail and Meyer [BIM95]. This restricts rules in the operational semantics to be "well-behaved". First-order abstract GSOS, which was introduced by Turi and Plotkin [TP97], does so by enforcing that the semantic rules correspond to a certain natural transformation. We will refer to their framework as "first-order", even though they did not use the prefix, to distinguish it from higher-order abstract GSOS (HO-GSOS), formulated by Goncharov et al. [Gon+24]. Like first-order GSOS, HO-GSOS models rules as certain natural transformations. Unlike the older framework, it allows for languages with "higher-order" behaviour, i.e. languages with name binding such as the λ-calculus and, as we will see, CCS. In both cases, by formulating the operational rules in the required way, one obtains an operational model with compositionality for free.

2.3.1 (First-Order) Abstract GSOS

The categorical framework of first-order abstract GSOS developed in [TP97] is a categorical theory that reformulates and generalizes the GSOS rule format found in [BIM95]. This, in turn, is based on the structured operational semantics (SOS) developed in [Plo04].

Definition 2.18. A *GSOS rule* is an operational rule of the form

$$\frac{\left(x_i \xrightarrow{\alpha} y_{i,j}^{\alpha}\right)_{1\leq i\leq m, 1\leq j\leq n_i^{\alpha}, \alpha\in A_i} \qquad \left(x_i \overset{\beta}{\not\to}\right)_{1\leq i\leq m, \beta\in B_i}}{f(x_1,\ldots,x_m) \xrightarrow{\gamma} t} \tag{2.22}$$

where all variables are distinct, $f \in \overline{\Sigma}$ is an operation of arity m, A_i and B_i are subsets of some set of labels $\mathcal{A}$ that also contains γ, and t is a term built over variables $x_i, y_{i,j}^{\beta}$.

The "abstractness" of Turi and Plotkin's work then comes in by realizing that rules in the GSOS format are in bijective correspondence to natural transformations of a specific form. In order to talk categorically about operational semantics, one first has to "categorify" the syntax of the discussed language. This is done via a signature functor $\Sigma\colon \mathcal{C} \to \mathcal{C}$, as shown in Equation (2.8). Another endofunctor $B\colon \mathcal{C} \to \mathcal{C}$ is needed to capture the behaviour of the language.

A GSOS specification, i.e. a set of GSOS rules, then corresponds to a natural transformation ϱ, called a GSOS law:

$$\varrho\colon \Sigma(\mathrm{Id} \times B) \Rightarrow B\Sigma^{\star} \tag{2.23}$$

where $\Sigma^{\star}\colon \mathcal{C} \to \mathcal{C}$ is the free monad induced by Σ.

From such GSOS laws we can obtain both an operational model as well as a denotational model. They occur as ϱ-bialgebras, i.e. a Σ-algebra $(X, a\colon \Sigma X \to X)$ together with a B-coalgebra $(X, c\colon X \to BX)$ such that the diagram commutes:

$$\begin{tikzcd}
\Sigma X \arrow[r, "a"] \arrow[d, "\Sigma\langle \mathrm{id}_X, c\rangle"] & X \arrow[r, "c"] & BX \\
\Sigma(X \times BX) \arrow[rr, "\varrho_X"] & & B\Sigma^{\star} X \arrow[u, "B\hat{a}"']
\end{tikzcd} \tag{2.24}$$

where $\hat{a}\colon \Sigma^{\star} X \to X$ is the Eilenberg-Moore algebra induced by a.

One such ϱ-bialgebra can be constructed by initiality of $(\mu\Sigma, \iota)$, which gives a B-coalgebra $\gamma\colon \mu\Sigma \to B(\mu\Sigma)$. This is called the *operational model* of ϱ and is the initial ϱ-bialgebra. Dually, we obtain another ϱ-bialgebra if B has a final coalgebra $(\nu B, \tau)$. This is called the *denotational model* of ϱ and is the final ϱ-bialgebra. By initiality and finality we get a unique ϱ-bialgebra morphism $[\![-]\!]_{\varrho}\colon \mu\Sigma \to \nu B$ that sends a closed term to its abstract behaviour:

$$\begin{tikzcd}
\Sigma\,\mu\Sigma \arrow[r, "\iota"] \arrow[d, "\Sigma[\![-]\!]_{\varrho}"] & \mu\Sigma \arrow[r, "\gamma"] \arrow[d, dashed, "[\![-]\!]_{\varrho}"] & B\,\mu\Sigma \arrow[d, "B[\![-]\!]_{\varrho}"] \\
\Sigma\,\nu B \arrow[r, "\alpha"] & \nu B \arrow[r, "\tau"] & B\,\nu B
\end{tikzcd} \tag{2.25}$$

Compositionality of this semantics is entailed immediately by initiality of the ϱ-bialgebra on $\mu\Sigma$ and finality of the ϱ-bialgebra on νB.

Strong bisimulation can then be constructed by taking the pullback of $[\![-]\!]_{\varrho}$ with itself. This gives us a semantics of the language that is fully abstract with respect to bisimulation, at least if B preserves weak pullbacks. Since this semantics is compositional, our notion of strong bisimulation is a Σ-congruence.

In the following we will formulate the operational semantics of a simple first-order language, namely CCS without fixpoints, in the first-order abstract GSOS to illustrate the usage of the framework and relate it to the formulation of CCS *with* fixpoints in Chapter 3.

Example 2.19 (CCS without fixpoints)**.** Consider the operational semantics of CCS given in Figure 2.2 but without the FIX rule. We can encode these rules using first-order abstract GSOS. Firstly, we need to give a categorical formulation of the syntax by means of a signature functor

$$\Sigma X = \coprod_{f \in \overline{\Sigma}} X^{\mathrm{ar}(f)} \tag{2.26}$$

where $\overline{\Sigma} \coloneqq \mathbb{N} \cup \{\varnothing, +, |\} \cup \{\mathrm{act}_\alpha \mid \alpha \in \mathcal{A}\} \cup \{\mathrm{restr}_L \mid L \subseteq \mathcal{A}\} \cup \{\mathrm{ren}_\varphi \mid \varphi \in \mathsf{Ren}(\mathcal{A})\}$. The arity function $\mathrm{ar} \colon \overline{\Sigma} \to \mathbb{N}$ is then given by

$$\mathrm{ar}(f) = \begin{cases} 0 & f \in \mathbb{N} \cup \{\varnothing\} \\ 1 & f \in \{\mathrm{act}_\alpha \mid \alpha \in \mathcal{A}\} \cup \{\mathrm{restr}_L \mid L \subseteq \mathcal{A}\} \cup \{\mathrm{ren}_\varphi \mid \varphi \in \mathsf{Ren}(\mathcal{A})\} \\ 2 & f \in \{+, |\} \end{cases} \tag{2.27}$$

To match the syntax of CCS more closely, we will afford ourselves some notational liberty, e.g. writing $\alpha \centerdot P$ to denote $\mathrm{in}_{\mathrm{act}_\alpha}(P)$.

Now that we have the syntax of CCS without fixpoints in categorical terms, we need to tackle their behaviour. As discussed in Section 2.2.1, the semantics of CCS processes can be described with labelled transition systems. Therefore, we need to find an endofunctor B such that LTS's over some set $\mathcal{A}$ are the final coalgebra νB. It is well-known that $BX = \mathcal{P}(\mathcal{A} \times X)$ serves that purpose. Since we only deal with binary sums and therefore countable branching in the LTS, we can instead use the countable powerset functor:

$$BX = \mathcal{P}_{\omega 1}(\mathcal{A} \times X) \tag{2.28}$$

The operational rules in Figure 2.2 (without the FIX rule) now give rise to a family $(\varrho_X)_{X \in \mathsf{Set}}$ of maps:

$$\begin{aligned}
&\varrho_X \colon \Sigma(X \times BX) \to B\Sigma^\star X \\
&\varrho_X(\varnothing) = \emptyset \\
&\varrho_X(\#m) = \emptyset \\
&\varrho_X(\alpha \centerdot (P, b_P)) = \{(\alpha, \eta_X(P))\} \\
&\varrho_X((P, b_P) + (Q, b_Q)) = \{(\alpha, \eta_X(P')) \mid (\alpha, P') \in b_P\} \cup \{(\alpha, \eta_X(Q')) \mid (\alpha, Q') \in b_Q\} \\
&\varrho_X((P, b_P) | (Q, b_Q)) = \quad \{(\alpha, (\eta_X(P')) | (\eta_X(Q))) \mid (\alpha, P') \in b_P\} \\
&\qquad\qquad \cup \{(\alpha, (\eta_X(P)) | (\eta_X(Q'))) \mid (\alpha, Q') \in b_Q\} \\
&\qquad\qquad \cup \left\{(\tau, (\eta_X(P')) | (\eta_X(Q'))) \,\middle|\, (\alpha, P') \in b_P, (\beta, Q') \in b_Q, \alpha = \overline{\beta}\right\} \\
&\varrho_X((P, b_P) \setminus L) = \{(\alpha, (\eta_X(P')) \setminus L) \mid (\alpha, P') \in b_P, \alpha, \overline{\alpha} \notin L\} \\
&\varrho_X(P\,[\varphi]) = \{(\varphi(\alpha), \eta_X(P')\,[\varphi]) \mid (\alpha, P') \in b_P\}
\end{aligned}$$

The framework of first-order abstract GSOS requires that the family $(\varrho_X)_{x \in \mathsf{Set}}$ forms a natural transformation, so let us prove that it is the case:

Proposition 2.20. $\varrho \colon \Sigma(\mathrm{Id} \times B) \Rightarrow B\Sigma^\star$ *is a natural transformation with component morphisms*

ϱ_X, *i.e. the following diagram commutes for all* $f\colon X \to Y$*:*

$$\begin{tikzcd}
\Sigma(X \times B(X)) \arrow[r, "\varrho_X"] \arrow[d, "\Sigma(f \times B(f))"] & B(\Sigma^\star(X)) \arrow[d, "B\Sigma^\star(f)"] \\
\Sigma(Y \times B(Y)) \arrow[r, "\varrho_Y"] & B(\Sigma^\star(Y))
\end{tikzcd} \tag{2.29}$$

Proof. Let $f\colon X \to Y$. The proof proceeds on elements by case distinction on $x \in \Sigma(X \times B\,X)$:

If $x = \varnothing$**, then** $(B\Sigma^\star(f))(\varrho_X(\varnothing)) = (B\Sigma^\star(f))(\emptyset) = \emptyset = \varrho_Y(\varnothing) = \varrho_Y(\Sigma(f \times B(f))(\varnothing))$.

If $x = \alpha\bullet(P, b_P)$**, then**

$$\begin{aligned}
&(B\Sigma^\star(f))(\varrho_X(\alpha\bullet(P, b_P)))\\
&= (B\Sigma^\star f)(\{(\alpha, \eta_X(P))\})\\
&= \{(\alpha, \eta_Y(f(P)))\}\\
&= \varrho_Y(\alpha\bullet(f(P), B(f)(b_P)))\\
&= \varrho_Y(\Sigma(f \times B(f))(\alpha\bullet(P, b_P)))
\end{aligned}$$

If $x = (P, b_P) + (Q, b_Q)$**, then**

$$\begin{aligned}
&(B\Sigma^\star(f))(\varrho_X((P, b_P) + (Q, b_Q)))\\
&= (B\Sigma^\star(f))(\{(\alpha, \eta_X(P')) \mid (\alpha, P') \in b_P\} \cup \{(\alpha, \eta_X(Q')) \mid (\alpha, Q') \in b_Q\})\\
&= \ \{(\alpha, \eta_Y(f(P'))) \mid (\alpha, P') \in (b_P)\} \cup \{(\alpha, \eta_Y(f(Q'))) \mid (\alpha, Q') \in (b_Q)\}\\
&= \ \{(\alpha, \eta_Y(P')) \mid (\alpha, P') \in B(f)(b_P)\} \cup \{(\alpha, \eta_Y(Q')) \mid (\alpha, Q') \in B(f)(b_Q)\}\\
&= \varrho_Y((f(P), B(f)(b_P)) + (f(Q), B(f)(b_Q)))\\
&= \varrho_Y(\Sigma(f \times B(f))((P, b_P) + (Q, b_Q)))
\end{aligned}$$

If $x = (P, b_P)|(Q, b_Q)$**, then**

$$\begin{aligned}
&(B\Sigma^\star(f))(\varrho_X((P, b_P)|(Q, b_Q)))\\
&= (B\Sigma^\star(f))\,(\{(\alpha, (\eta_X(P'))|(\eta_X(Q))) \mid (\alpha, P') \in b_P\}\\
&\qquad\qquad \cup \{(\alpha, (\eta_X(P))|(\eta_X(Q'))) \mid (\alpha, Q') \in b_Q\}\\
&\qquad\qquad \cup \Big\{(\tau, (\eta_X(P'))|(\eta_X(Q'))) \,\Big|\, (\alpha, P') \in b_P, (\beta, Q') \in b_Q, \alpha = \overline{\beta}\Big\}\Big)\\
&= \ \{(\alpha, (\eta_Y(f(P')))|(\eta_Y(f(Q)))) \mid (\alpha, P') \in b_P\}\\
&\quad \cup \{(\alpha, (\eta_Y(f(P)))|(\eta_Y(f(Q')))) \mid (\alpha, Q') \in b_Q\}\\
&\quad \cup \Big\{(\tau, (\eta_Y(f(P')))|(\eta_Y(f(Q')))) \,\Big|\, (\alpha, P') \in b_P, (\beta, Q') \in b_Q, \alpha = \overline{\beta}\Big\}\\
&= \ \{(\alpha, (\eta_Y(P'))|(\eta_Y(Q))) \mid (\alpha, P') \in B(f)(b_P)\}\\
&\quad \cup \{(\alpha, (\eta_Y(P))|(\eta_Y(Q'))) \mid (\alpha, Q') \in B(f)(b_Q)\}\\
&\quad \cup \Big\{(\tau, (\eta_Y(P'))|(\eta_Y(Q'))) \,\Big|\, (\alpha, P') \in B(f)(b_P), (\beta, Q') \in B(f)(b_Q), \alpha = \overline{\beta}\Big\}\\
&= \varrho_Y((f(P), B(f)(b_P))|(f(Q), B(f)(b_Q)))\\
&= \varrho_Y(\Sigma(f \times B(f))((P, b_P)|(Q, b_Q)))
\end{aligned}$$

If $x = (P, b_P) \setminus \varphi$, then

$$\begin{aligned}
&(B\Sigma^\star(f))(\varrho_X((P, b_P) \setminus L)) \\
&= (B\Sigma^\star(f))(\{(\alpha, (\eta_X(P')) \setminus L) \mid (\alpha, P') \in b_P, \alpha, \overline{\alpha} \notin L\}) \\
&= \{(\alpha, (\eta_Y(f(P'))) \setminus L) \mid (\alpha, P') \in b_P, \alpha, \overline{\alpha} \notin L\} \\
&= \{(\alpha, (\eta_Y(P')) \setminus L) \mid (\alpha, P') \in B(f)(b_P), \alpha, \overline{\alpha} \notin L\} \\
&= \varrho_Y((f(P), B(f)(b_P)) \setminus L) \\
&= \varrho_Y(\Sigma(f \times B(f))(P \setminus L))
\end{aligned}$$

If $x = (P, b_P)\,[\varphi]$, then

$$\begin{aligned}
&(B\Sigma^\star(f))(\varrho_X((P, b_P)\,[\varphi])) \\
&= (B\Sigma^\star(f))(\{(\varphi(\alpha), (\eta_X(P'))\,[\varphi]) \mid (\alpha, P') \in b_P\}) \\
&= \{(\varphi(\alpha), (\eta_Y(f(P')))\,[\varphi]) \mid (\alpha, P') \in b_P\} \\
&= \{(\varphi(\alpha), (\eta_Y(P'))\,[\varphi]) \mid (\alpha, P') \in B(f)(b_P)\} \\
&= \varrho_Y((f(P), B(f)(b_P))\,[\varphi]) \\
&= \varrho_Y(\Sigma(f \times B(f))(P\,[\varphi]))
\end{aligned}$$

□

By the previously outlined steps we can obtain from ϱ a compositional operational semantics with bisimulation as a congruence.

2.3.2 Higher-Order Abstract GSOS

In the previous section, we have seen how first-order abstract GSOS allows us to obtain a compositional operational model of a language with a suited operational semantics. However, we cannot encode "higher-order" languages, where we allow for programs to be passed as values. The HO-GSOS framework introduced by Goncharov et al. [Gon+24] remedies this restriction by giving a theory that provides compositionality results such as those obtained by first-order abstract GSOS. In this section, we will review the part of the framework that is relevant for the later development of this thesis.

Again, there is a rule format that specifies those rules that fit the framework:

$$\frac{(x_j \longrightarrow y_j)_{j \in W} \qquad \left(x_i \xrightarrow{z} y_i^z\right)_{i \in \overline{W}, z \in \{x_1, \ldots, x_n\}}}{f(x_1, \ldots, x_n) \longrightarrow t} \tag{2.30}$$

$$\frac{(x_j \longrightarrow y_j)_{j \in W} \qquad \left(x_i \xrightarrow{z} y_i^z\right)_{i \in \overline{W}, z \in \{x, x_1, \ldots, x_n\}}}{f(x_1, \ldots, x_n) \xrightarrow{x} t} \tag{2.31}$$

where $f \in \overline{\Sigma}$ is an operator of arity n, $W \subseteq \{1, \ldots, n\}$, $\overline{W} = \{1, \ldots, n\} \setminus W$ and t is a term built over variables $V \coloneqq \{x, x_i, y_i, y_i^{x_j}\}$, i.e. $t \in \Sigma^\star(V)$.

Rules of this rule format now correspond to families of maps

$$\varrho_{X,Y} \colon \Sigma(X \times B(X, Y)) \to B(X, \Sigma^\star(X + Y)) \tag{2.32}$$

which are dinatural in X and natural in Y, which we will call a HO-GSOS law.

A HO-GSOS law induces an operational model as a $B(\mu\Sigma, -)$-coalgebra $\iota^\clubsuit\colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$ given by initiality of $(\mu\Sigma, \iota)$:

$$\begin{tikzcd}
\Sigma\mu\Sigma \arrow[rr, "\iota"] \arrow[d, "\Sigma(\langle \mathrm{id}_{\mu\Sigma}, \iota^\clubsuit \rangle)"] & & \mu\Sigma \arrow[d, "\langle \mathrm{id}_{\mu\Sigma}, \iota^\clubsuit \rangle"] \\
\Sigma(\mu\Sigma \times B(\mu\Sigma, \mu\Sigma)) \arrow[r, "\langle \iota \circ \Sigma(\mathrm{fst}), \varrho_{\mu\Sigma,\mu\Sigma} \rangle"] & \mu\Sigma \times B(\mu\Sigma, \Sigma^\star(\mu\Sigma + \mu\Sigma)) \arrow[r, "\mathrm{id}_{\mu\Sigma} \times B(\mathrm{id}_{\mu\Sigma}, \hat{\iota} \circ \Sigma^\star(\nabla_{\mu\Sigma}))"] & \mu\Sigma \times B(\mu\Sigma, \mu\Sigma)
\end{tikzcd} \tag{2.33}$$

If $B(\mu\Sigma, -)$ has a final coalgebra, then we can obtain strong bisimulation via the following pullback:

$$\begin{tikzcd}
\sim \arrow[r, "\mathrm{fst}|_\sim"] \arrow[d, "\mathrm{snd}|_\sim"] \arrow[dr, phantom, very near start, "\lrcorner"] & \mu\Sigma \arrow[d, "\mathbf{coit}\,\iota^\clubsuit"] \\
\mu\Sigma \arrow[r, "\mathbf{coit}\,\iota^\clubsuit"] & \nu\gamma.\, B(\mu\Sigma, \gamma)
\end{tikzcd} \tag{2.34}$$

By compositionality, this strong bisimulation is a congruence with respect to Σ.

3 CCS in Higher-Order Abstract GSOS

This section contains the encoding of guarded CCS *with* fixpoints in the framework of HO-GSOS. For the simpler case of CCS without fixpoints, see Example 2.19, where we give the formulation in first-order abstract GSOS. From Section 3.1 to Section 3.3 we will see how one can reformulate the standard operational semantics of CCS given in Figure 2.2 to obtain an equivalent semantics that fits into the rule format of HO-GSOS. In Section 3.4 this operational semantics is used to define a dinatural transformation ϱ which allows us to apply the HO-GSOS framework and obtain a compositional operational model of guarded CCS.

3.1 Semantics of the Fixpoint operator

Recall the standard semantics of the fixpoint operator given in Figure 2.2:

$$\frac{P\,[0 \mapsto \text{fix } P] \xrightarrow{\alpha} P'}{\text{fix } P \xrightarrow{\alpha} P'}\ \textsc{fix} \tag{3.1}$$

This does not fit into the rule format of HO-GSOS. In particular, we cannot encode the term $P\,[0 \mapsto \text{fix } P]$ containing a substitution in the left-hand side of the premise since it is not a subterm of fix P in the conclusion. Therefore, the standard operational semantics is not directly suitable for encoding in HO-GSOS.

Let us therefore consider an alternative rule for the fixpoint (`Fix'`):

$$\frac{P \xrightarrow{\alpha} P'}{\text{fix } P \xrightarrow{\alpha} P'\,[0 \mapsto \text{fix } P]}\ \textsc{fix'} \tag{3.2}$$

The main idea is to move the substitution down into the right-hand side of the premise, where the HO-GSOS rule format is more lenient. To illustrate that this move is in fact valid, consider this example derivation:

Example 3.1 (Revisiting Example 2.9). Let us reconsider the term $P \coloneqq \text{fix } (\text{fix } \alpha \bullet \#\mathit{1})$. We can show that in the alternative fixpoint semantics, P also takes an α-step to itself:

$$\cfrac{\cfrac{\cfrac{}{\alpha \bullet \#\mathit{1} \xrightarrow{\alpha}_{\text{fix}'} \#\mathit{1}}\ \textsc{act}}{\text{fix } \alpha \bullet \#\mathit{1} \xrightarrow{\alpha}_{\text{fix}'} (\#\mathit{1}\,[0 \mapsto \text{fix } \alpha \bullet \#\mathit{1}]) = \#\mathit{0}}\ \textsc{fix'}}{\text{fix fix } \alpha \bullet \#\mathit{1} \xrightarrow{\alpha}_{\text{fix}'} (\#\mathit{0}\,[0 \mapsto \text{fix fix } \alpha \bullet \#\mathit{1}]) = \text{fix fix } \alpha \bullet \#\mathit{1}}\ \textsc{fix'}$$

To show that one may use both versions of the fixpoint rule interchangeably, we first need to examine how substitutions interact with the standard step relation. In the following, let $\longrightarrow_{\text{fix}'}$ denote the operational semantics with the alternative rule for the fixpoint.

First, let us show that we can apply a substitution to both sides of the step relation in the standard operational semantics:

Lemma 3.2 (step-subst). *Let P, P' be processes, σ a substitution. If P takes an α-step to P' then $P\sigma$ takes an α-step to $P'\sigma$:*

$$P \xrightarrow{\alpha} P' \implies P\sigma \xrightarrow{\alpha} P'\sigma$$

Proof. By induction over the derivation. The cases for non-fixpoint steps are trivial. In the FIX case we get

$$\dfrac{P\,[0 \mapsto \text{fix } P] \xrightarrow{\alpha} P'}{\text{fix } P \xrightarrow{\alpha} P'}\ \text{FIX}$$

Now by the induction hypothesis we have $(P\,[0 \mapsto \text{fix } P])\ \sigma \xrightarrow{\alpha} P'\ \sigma$. By Theorem 2.8 we get $(P\,[0 \mapsto \text{fix } P])\ \sigma = (P\ (\uparrow\ \sigma))\,[0 \mapsto \text{fix } (P\ (\uparrow\ \sigma))]$ and thus

$$\dfrac{(P\ (\uparrow\ \sigma))\,[0 \mapsto \text{fix } (P\ (\uparrow\ \sigma))]}{(\text{fix } P)\ \sigma = \text{fix } (P\ (\uparrow\ \sigma)) \xrightarrow{\alpha} P'\ \sigma}\ \text{FIX} \qquad \square$$

Now we can state the equivalence of the two operational semantics. This equivalence comes with a caveat: It only holds if the process taking a step is guarded. To see why this is the case, consider this counterexample:

Example 3.3 (An unguarded term). Let $P \coloneqq \text{fix } (\#\mathit{0}|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing)$. P is an unguarded term, since the variable $\#\mathit{0}$ is not prefixed by any action. Nevertheless, one can show that P takes a τ-step in the standard semantics:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha\bullet\varnothing \xrightarrow{\alpha} \varnothing}\ \text{ACT}}{\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing \xrightarrow{\alpha} \varnothing}\ \text{SUM}_\text{l}}{P|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing \xrightarrow{\alpha} P|\varnothing}\ \text{PAR}_\text{r}}{\text{fix } (\#\mathit{0}|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing) \xrightarrow{\alpha} P|\varnothing}\ \text{FIX} \qquad \dfrac{\dfrac{}{\overline{\alpha}\bullet\varnothing \xrightarrow{\overline{\alpha}} \varnothing}\ \text{ACT}}{\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing \xrightarrow{\overline{\alpha}} \varnothing}\ \text{SUM}_\text{r}}{P|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing \xrightarrow{\tau} (P|\varnothing)|\varnothing}\ \text{SYNC}}{\text{fix } (\#\mathit{0}|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing) \xrightarrow{\tau} (P|\varnothing)|\varnothing}\ \text{FIX}$$

Trying to derive the same τ-step in the alternative fixpoint semantics fails: The unguarded variable on the left-hand side of the parallel composition cannot take a step:

$$\dfrac{\dfrac{\#\mathit{0} \nrightarrow \qquad \dfrac{\dfrac{}{\alpha|\bullet\varnothing \xrightarrow{\overline{\alpha}} \varnothing}\ \text{FIX'}}{\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing \xrightarrow{\overline{\alpha}} \varnothing}\ \text{SUM}_\text{r}}{\#\mathit{0}|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing \xrightarrow{\tau} (P|\varnothing)|\varnothing}\ \text{SYNC}}{\text{fix } (\#\mathit{0}|\alpha\bullet\varnothing + \overline{\alpha}\bullet\varnothing) \xrightarrow{\tau}_{\text{fix}'} (P|\varnothing)|\varnothing}\ \text{FIX'}$$

This shows that for an unguarded process like P the standard semantics allows behaviour which our alternative semantics cannot. Thus we will restrict ourselves to guarded processes in the following.

Let us now prove a property relating guarded terms with substitutions in the standard operational semantics that we are going to need later on:

Lemma 3.4 (subst-fix-swap). *Let P be a guarded term, T and P' arbitrary terms. If $P\,[0 \mapsto T] \xrightarrow{\alpha} P'$ then there exists a term Q such that $P \xrightarrow{\alpha} Q$ and $P' = Q\,[0 \mapsto T]$.*

Proof. We proceed by simultaneous induction over the derivation of $P\,[0 \mapsto T] \xrightarrow{\alpha} P'$ and over P itself.

If $P = \#n$**, then** P is not guarded and the case holds vacuously.

In the remaining non-fixpoint cases the goal follows directly:

If $P = R|S$ **and** $R|S \xrightarrow{\tau} R'|S'$**, then** since P is guarded, so are R and S. We can therefore apply the induction hypothesis to both and obtain terms R'' and S'' such that $R \xrightarrow{\alpha} R''$, $R' = R''\,[0 \mapsto T]$ and $S \xrightarrow{\overline{\alpha}} S''$, $S' = S''\,[0 \mapsto T]$. Then there exists $R''|S''$ such that

$$\frac{R'' \xrightarrow{\alpha} R' \qquad S'' \xrightarrow{\alpha} S'}{R''|S'' \xrightarrow{\tau} R'|S'}\ \text{SYNC}$$

and $R'|S' = (R''|S'')\,[0 \mapsto T]$ holds directly from the above equalities.

If $P = \text{fix } R$ **and** $(\text{fix } R)\,[0 \mapsto T] = \text{fix } (R\,[1 \mapsto T]) \xrightarrow{\alpha} P'$**, then** we know that since P is guarded, so is R. By Lemma 2.16 we know that $R\,[0 \mapsto \text{fix } R]$ is guarded. Furthermore, we can obtain

$$\frac{(R\,[1 \mapsto T])\,[0 \mapsto \text{fix } (R\,[1 \mapsto T])] \xrightarrow{\alpha} P'}{\text{fix } (R\,[1 \mapsto T]) \xrightarrow{\alpha} P'}\ \text{FIX}$$

From Theorem 2.8 we know that

$$(R\,[1 \mapsto T])\,[0 \mapsto \text{fix } (R\,[1 \mapsto T])] = (R\,[0 \mapsto \text{fix } R])\,[0 \mapsto T]$$

With this we can apply the induction hypothesis to obtain a term Q such that $R\,[0 \mapsto \text{fix } R] \xrightarrow{\alpha} Q$ and therefore

$$\frac{R\,[0 \mapsto \text{fix } R] \xrightarrow{\alpha} Q}{\text{fix } R \xrightarrow{\alpha} Q}\ \text{FIX}$$ □

Remark 3.5. If we left out the guardedness assumption, the proposition above would not hold: Consider the case where $P := \#0$ and $T := \alpha \centerdot \varnothing$. Then clearly $P\,[0 \mapsto T] = \alpha \centerdot \varnothing \xrightarrow{\alpha} \varnothing$, but there cannot exist a term Q such that $P \xrightarrow{\alpha} Q$.

Now we can state the main equivalence:

Theorem 3.6 (fix⇔fix'). *For guarded terms, the alternative fixpoint semantics is equivalent to the standard semantics.*

$$\mathsf{guarded}(P) \implies (P \xrightarrow{\alpha} P' \iff P \xrightarrow{\alpha}_{\text{fix}'} P')$$

for all $\alpha \in \mathcal{A}$, $P, P' \in$ Proc

Proof. We prove the two directions of the equivalence separately:

"$\Longrightarrow$" We proceed by induction over the derivation of $P \xrightarrow{\alpha} P'$. The non-fixpoint cases are trivial, since the two semantics share those rules. In the fixpoint case, we assume

$$\frac{P\,[0 \mapsto \text{fix } P] \xrightarrow{\alpha} P'}{\text{fix } P \xrightarrow{\alpha} P'}\ \text{FIX}$$

We can now apply Lemma 3.4 to obtain a term Q such that $P \xrightarrow{\alpha} Q$ and $P' = Q\,[0 \mapsto \text{fix } P]$. By the induction hypothesis we get $P \xrightarrow{\alpha}_{\text{fix}'} Q$ and thus

$$\dfrac{P \xrightarrow{\alpha}_{\text{fix}'} Q}{\text{fix } P \xrightarrow{\alpha}_{\text{fix}'} Q\,[0 \mapsto \text{fix } P]}\ \text{FIX'}$$

By the above equality we finally obtain $\text{fix } P \xrightarrow{\alpha}_{\text{fix}'} P'$.

"$\Longleftarrow$" Here we induct over the fix$'$-derivation. Again, all cases except the FIX$'$-step are trivial. In the remaining case, we assume

$$\dfrac{P \xrightarrow{\alpha}_{\text{fix}'} P'}{\text{fix } P \xrightarrow{\alpha}_{\text{fix}'} P'\,[0 \mapsto \text{fix } P]}\ \text{FIX'}$$

By the induction hypothesis we get $P \xrightarrow{\alpha} P'$, which we can apply to Lemma 3.2 with $\sigma := [0 \mapsto \text{fix } P]$ to get

$$\dfrac{P\,[0 \mapsto \text{fix } P] \xrightarrow{\alpha} P'\,[0 \mapsto \text{fix } P]}{\text{fix } P \xrightarrow{\alpha} P'\,[0 \mapsto \text{fix } P]}\ \text{FIX} \qquad \square$$

This equivalence allows us to avoid the problematic substitution in the premise, at least if we restrict to guarded terms. We are not out of the woods yet: Recall that in the HO-GSOS format, the right-hand side of the conclusion should be in $\Sigma^\star(X + Y)$ for some objects X and Y, which is not the case for $P'\,[0 \mapsto \text{fix } P]$ in our current rule. We will therefore need to move the substitution again, so that it finally fits the HO-GSOS format. Our efforts here were not in vain, since we will be using the alternative fixpoint semantics as a intermediate when proving the equivalence of a new semantics to the standard semantics.

3.2 Labelling with Substitutions

In the previous section we have seen how the occurrences of substitutions in the operational semantics that are necessary to define the behaviour of the fixpoint operator are problematic. Even after moving the substitution from the premise into the conclusion, it is still not clear how this translates into the categorical framework of HO-GSOS. A position that has remained untouched so far presents itself now: Instead of labelling steps just by actions from a set $\mathcal{A}$, we can add a substitution that should be applied to the resulting term.

We will denote this new operational semantics as $\xrightarrow{\alpha}_{\sigma}$ where $\alpha \in \mathcal{A}$ and σ is a substitution. Since we have changed the set of labels, we need a different notion of equivalence between the standard semantics and this new one:

$$(\exists\, P'.\, P \xrightarrow{\alpha} P' \wedge P'' = P'\,\sigma) \iff P \xrightarrow{\alpha}_{\sigma} P'' \tag{3.3}$$

Figure 3.1 lays out the rules for this new operational semantics. Most non-fixpoint rules behave analogously to the standard semantics, except when a term from the left side is reused on the right side. In these cases, the substitution needs to be applied to the term. This happens in the ACT rule, where σ is applied to P and in the PAR$_\text{l}$ and PAR$_\text{r}$ rules. There one can see that σ is only applied respectively to the right and left sides of the parallel composition, i.e. the terms that do not take a step in their respective rules.

Let us illustrate how this new operational semantics works:

$$\dfrac{P' = P\sigma}{\alpha \bullet P \xrightarrow{\alpha}_{\sigma} P'}\ \text{ACT}$$

$$\dfrac{P \xrightarrow{\alpha}_{\sigma} P'}{P + Q \xrightarrow{\alpha}_{\sigma} P'}\ \text{SUM}_{\text{l}} \qquad \dfrac{Q \xrightarrow{\alpha}_{\sigma} Q'}{P + Q \xrightarrow{\alpha}_{\sigma} Q'}\ \text{SUM}_{\text{r}}$$

$$\dfrac{P \xrightarrow{\alpha}_{\sigma} P'}{P|Q \xrightarrow{\alpha}_{\sigma} P'|Q\,\sigma}\ \text{PAR}_{\text{l}} \qquad \dfrac{Q \xrightarrow{\alpha}_{\sigma} Q'}{P|Q \xrightarrow{\alpha}_{\sigma} P\,\sigma|Q'}\ \text{PAR}_{\text{r}}$$

$$\dfrac{P \xrightarrow{\alpha}_{\sigma} P' \qquad Q \xrightarrow{\overline{\alpha}}_{\sigma} Q'}{P|Q \xrightarrow{\tau}_{\sigma} P'|Q'}\ \text{SYNC}$$

$$\dfrac{P \xrightarrow{\alpha}_{\sigma} P' \qquad \alpha, \overline{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha}_{\sigma} P'}\ \text{RES} \qquad \dfrac{P \xrightarrow{\alpha}_{\sigma} P'}{P\,[\varphi] \xrightarrow{\varphi(\alpha)}_{\sigma} P'\,[\varphi]}\ \text{REN}$$

$$\dfrac{P \xrightarrow{\alpha}_{(\sigma \circ [0 \mapsto \text{fix } \text{P}])} P'}{\text{fix } P \xrightarrow{\alpha}_{\sigma} P'}\ \text{FIX}$$

Figure 3.1: Operational Semantics labelled with substitutions
(Step.SubstitutionLabels)

Example 3.7 (Revisiting Example 2.9, again)**.** We can show that the term $P :=$ fix (fix $\alpha \bullet \#1$) takes an α-step to itself, as we did in Examples 2.9 and 3.1:

$$\dfrac{\dfrac{\dfrac{\overline{(\#1\ [0 \mapsto \text{fix } \alpha \bullet \#1])\ [0 \mapsto \text{fix fix } \alpha \bullet \#1] = \#0\ [0 \mapsto \text{fix fix } \alpha \bullet \#1] = \text{fix fix } \alpha \bullet \#1}}{\alpha \bullet \#1 \xrightarrow{\alpha}_{[0 \mapsto \text{fix fix } \alpha \bullet \#1] \circ [0 \mapsto \text{fix } \alpha \bullet \#1]} \text{fix fix } \alpha \bullet \#1}\ \text{ACT}}{\text{fix } \alpha \bullet \#1 \xrightarrow{\alpha}_{[0 \mapsto \text{fix fix } \alpha \bullet \#1]} \text{fix fix } \alpha \bullet \#1}\ \text{FIX}}{\text{fix fix } \alpha \bullet \#1 \xrightarrow{\alpha}_{\text{id}} \text{fix fix } \alpha \bullet \#1}\ \text{FIX}$$

Now let us prove that the equivalence holds for the alternative fixpoint semantics:

Theorem 3.8 (subst-step⇔fix')**.** *The substitution-labelled operational semantics is equivalent to the alternative fixpoint semantics:*

$$(\exists\, P'.\, P \xrightarrow{\alpha}_{\text{fix}'} P' \wedge P'' = P'\,\sigma) \iff P \xrightarrow{\alpha}_{\sigma} P''$$

for all $\alpha \in \mathcal{A}$, $P, P'' \in \text{Proc}$, $\sigma \in \mathsf{Subst}$.

Proof. We consider both directions of the equivalence separately:

"$\Longrightarrow$" Let P' be a process such that $P \xrightarrow{\alpha}_{\text{fix}'} P'$ and $P'' = P'\sigma$. We proceed by induction over the derivation of $P \xrightarrow{\alpha}_{\text{fix}'} P'$:

If $P = \alpha \bullet P'$**, then** P takes an ACT step:

$$\dfrac{}{\alpha \bullet Q \xrightarrow{\alpha}_{\text{fix}'} P'}\ \text{ACT}$$

Since $P'' = P'\sigma$, we need to show that P takes an α-step to P'' in the substitution-labelled semantics, which it does by:

$$\frac{P'' = P'\sigma}{\alpha\bullet P' \xrightarrow{\alpha}_{\sigma} P''}\ \textsc{act}$$

If $P = R|T$ and P takes a $\textsc{par}_\text{l}$ step, then

$$\frac{R \xrightarrow{\alpha}_{\text{fix}'} R'}{(R|T) \xrightarrow{\alpha}_{\text{fix}'} (R'|T) = P'}\ \textsc{par}_\text{l}$$

Since $P'' = P'\sigma = (R'\sigma)|(T\sigma)$, we can apply the induction hypothesis to obtain $R \xrightarrow{\alpha}_{\sigma} R'\sigma$ and therefore:

$$\frac{R \xrightarrow{\alpha}_{\sigma} R'\sigma}{R|T \xrightarrow{\alpha}_{\sigma} R'\sigma|(T\sigma) = P''}\ \textsc{par}_\text{l}$$

The remaining non-fixpoint cases are follow analogously, so let us move on:

If $P = \text{fix}\ Q$**, then** P takes a $\textsc{fix}'$ step:

$$\frac{Q \xrightarrow{\alpha}_{\text{fix}'} Q'}{\text{fix}\ Q \xrightarrow{\alpha}_{\text{fix}'} Q'\,[0 \mapsto \text{fix}\ Q] = P'}\ \textsc{fix'}$$

By the induction hypothesis, we get $Q \xrightarrow{\alpha}_{\sigma\circ[0\mapsto\text{fix Q}]} P'(\sigma \circ [0 \mapsto \text{fix}\ Q])$ and therefore:

$$\frac{Q \xrightarrow{\alpha}_{\sigma\circ[0\mapsto\text{fix Q}]} P'(\sigma \circ [0 \mapsto \text{fix}\ Q])}{\text{fix}\ Q \xrightarrow{\alpha}_{\sigma} P'(\sigma \circ [0 \mapsto \text{fix}\ Q]) = P''}\ \textsc{fix}$$

"$\Longleftarrow$" This direction follows immediately by induction over the derivation of $P \xrightarrow{\alpha}_{\sigma} P''$. □

Corollary 3.9 (subst-step⇔fix). *For guarded processes, the standard operational semantics is equivalent to the substitution-labelled operational semantics.*

$$(\exists\, P'.\, P \xrightarrow{\alpha} P' \wedge P'' = P'\,\sigma) \iff P \xrightarrow{\alpha}_{\sigma} P''$$

for all $\alpha \in \mathcal{A}$, $P, P'' \in \text{Proc}$, $\sigma \in \mathsf{Subst}$.

Proof. This follows directly from Theorem 3.6 and Theorem 3.8. □

3.3 Labelling with Finite Term Sequences

We should clarify how substitutions are modelled categorically: One approach is to use objects of type

$$\coprod_{n\in\mathbb{N}} Y^{(\Sigma^\star X)^n} \tag{3.4}$$

where X and Y are sets of variables. This would model arbitrary substitutions. In our case however, we can get away with using a more restrictive model. In Figure 3.1, we can see that the $\textsc{fix}$ rule is the only rule modifying the labelled substitution. Then we compose with a substitution $[0 \mapsto \text{fix}\ P]$. This means that *all* substitutions that occur in labels are the composition of substitutions of that shape.

We can model these substitutions in a simpler way as objects of type

$$Y^{(1+X)^*} \tag{3.5}$$

We will use $\varepsilon \in (1+X)^*$ to denote the empty sequence. For a sequence ξ and element $x \in 1+X$, we denote by $x :: \xi$ the sequence given by adding x to the beginning of ξ. Since we use the left summand of $1+X$ to designate a "hole" in the substitution, let us use some suggestive notation by denoting $\Box \coloneqq \mathrm{inl}(*)$.

Figure 3.2 illustrates how such a substitution is applied to the terms of CCS. Notice how in the NAME$_1$ rule, we wrap the variable P_i obtained from ξ in a fixpoint operator.

$$\frac{P' = P[\xi]}{(\alpha \bullet P)[\xi] = \alpha \bullet P'}\ \textsc{ACT}_{\textsc{SUB}}$$

$$\frac{P_i \neq \Box}{\#i\,[P_0, P_1, \ldots, P_n] = \text{fix } P_i}\ \textsc{NAME}_1 \qquad \frac{P_i = \Box}{\#i\,[P_0, P_1, \ldots, P_n] = \#i}\ \textsc{NAME}_2$$

$$\frac{P[\xi] = P' \qquad Q[\xi] = Q'}{(P+Q)[\xi] = P' + Q'}\ \textsc{SUM}_{\textsc{SUB}} \qquad \frac{P[\xi] = P' \qquad Q[\xi] = Q'}{(P|Q)[\xi] = P'|Q'}\ \textsc{PAR}_{\textsc{SUB}}$$

$$\frac{P[\xi] = P'}{(P \setminus L)[\xi] = P' \setminus L}\ \textsc{RES}_{\textsc{SUB}} \qquad \frac{P[\xi] = P'}{(P\,[\varphi])[\xi] = P'\,[\varphi]}\ \textsc{REN}_{\textsc{SUB}}$$

$$\frac{P[\Box, \xi] = P'}{(\text{fix } P)[\xi] = \text{fix } P'}\ \textsc{FIX}_{\textsc{SUB}}$$

Figure 3.2: Semantics of term sequence application

The operational semantics labelled with term sequences can be found in Figure 3.3. Observe that this mirrors the operational semantics labelled with substitutions from Figure 3.1 very closely.

Theorem 3.10. *The operational semantics labelled with finite term sequences is equivalent to the labelling with substitutions.*

Proof. This follows directly since the rules only differ in the type of substitutions. $\Box$

Then finally we can state that this operational semantics is equivalent to the standard semantics:

Corollary 3.11. *For guarded processes, the operational semantics labelled with finite term sequences is equivalent to the standard operational semantics.*

Proof. This follows from Corollary 3.9 and Theorem 3.10. $\Box$

3.4 Abstract Modeling of the Semantics

Now that we have defined an operational semantics that fits the HO-GSOS rule format, we can proceed with modelling this semantics categorically. Recall from Equation (2.32), that a

$$\frac{P' = P[\xi]}{\alpha \bullet P \xrightarrow{\alpha}_{\xi} P'}\;\text{ACT}$$

$$\frac{P \xrightarrow{\alpha}_{\xi} P'}{P + Q \xrightarrow{\alpha}_{\xi} P'}\;\text{SUM}_{\text{l}} \qquad \frac{Q \xrightarrow{\alpha}_{\xi} Q'}{P + Q \xrightarrow{\alpha}_{\xi} Q'}\;\text{SUM}_{\text{r}}$$

$$\frac{P \xrightarrow{\alpha}_{\xi} P'}{P|Q \xrightarrow{\alpha}_{\xi} P'|Q[\xi]}\;\text{PAR}_{\text{l}} \qquad \frac{Q \xrightarrow{\alpha}_{\xi} Q'}{P|Q \xrightarrow{\alpha}_{\xi} P[\xi]|Q'}\;\text{PAR}_{\text{r}}$$

$$\frac{P \xrightarrow{\alpha}_{\xi} P' \qquad Q \xrightarrow{\overline{\alpha}}_{\xi} Q'}{P|Q \xrightarrow{\tau}_{\xi} P'|Q'}\;\text{SYNC}$$

$$\frac{P \xrightarrow{\alpha}_{\xi} P' \qquad \alpha, \overline{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha}_{\xi} P'}\;\text{RES} \qquad \frac{P \xrightarrow{\alpha}_{\sigma} P'}{P\,[\varphi] \xrightarrow{\varphi(\alpha)}_{\xi} P'\,[\varphi]}\;\text{REN}$$

$$\frac{P \xrightarrow{\alpha}_{\mathrm{P,P_0,\ldots,P_n}} P'}{\text{fix } P \xrightarrow{\alpha}_{\mathrm{P_0,\ldots,P_n}} P'}\;\text{FIX}$$

Figure 3.3: Operational Semantics labelled with term sequences

higher-order GSOS law is a family of maps

$$\varrho_{X,Y} \colon \Sigma(X \times B(X,Y)) \to B(X, \Sigma^{\star}(X+Y))$$

dinatural in X and natural in Y. To give meaning to this, we first need to define what the functors Σ and B are. We define Σ by the standard construction for signature functors as

$$\Sigma X = \coprod_{f \in \overline{\Sigma}} X^{\mathrm{ar}(f)} \tag{3.6}$$

where $\overline{\Sigma} \coloneqq \mathbb{N} \cup \{\varnothing, +, |, \mathrm{fix}\ \} \cup \{\mathrm{act}_{\alpha} \mid \alpha \in \mathcal{A}\} \cup \{\mathrm{restr}_{L} \mid L \subseteq \mathcal{A}\} \cup \{\mathrm{ren}_{\varphi} \mid \varphi \in \mathsf{Ren}(\mathcal{A})\}$.

In order for this to match the syntax of CCS more closely, we will again afford ourselves some notational liberty, e.g. writing $\alpha \bullet P$ to denote $\mathrm{in}_{\mathrm{act}_{\alpha}}(P)$.

With the syntax covered, we need to consider the behaviour functor. With the operational semantics labelled with term sequences, the behaviour of a process P consists of two parts: One that captures the behaviour of the term sequence label on P and another part that corresponds to the LTS obtained from applying the operational semantics rules on P. In the second part, we again need to label any process that P can take a step to with the behaviour of the labelled term sequence on that process. Our behaviour functor is therefore given by:

$$B(X,Y) = Y^{(1+X)^*} \times \mathcal{P}_{\omega 1}\left(\mathcal{A} \times Y^{(1+X)^*}\right) \tag{3.7}$$

The operational semantics given in Figure 3.3 then corresponds to a dinatural transformation

$$\varrho_{X,Y} \colon \Sigma(X \times B(X,Y)) \to B(X, \Sigma^{\star}(X+Y))$$

where $\varrho_{X,Y} \coloneqq \langle \varrho_{\text{l}}, \varrho_{\text{r}} \rangle$ (ϱ) is given by

$\varrho_{\text{l}} \colon \Sigma(X \times B(X,Y)) \to (\Sigma^{\star}(X+Y))^{(1+X)^*}$

$\varrho_{\mathrm{l}}(\varnothing) = \lambda\,\xi.\,\iota_{X+Y}(\varnothing)$

$\varrho_{\mathrm{l}}(\#m) = \lambda\,(P_1,\ldots,P_n).\begin{cases}\iota_{X+Y}(\text{fix }(\eta_{X+Y}(\mathrm{inl}(P)))) & m \leq n \wedge P_m = \mathrm{inl}(P)\\ \iota_{X+Y}(m) & \text{otherwise}\end{cases}$

$\varrho_{\mathrm{l}}(\alpha\,{\bullet}\,(P,\sigma,b)) = \lambda\,\xi.\,\iota_{X+Y}(\alpha\,{\bullet}\,\eta_{X+Y}(\mathrm{inr}(\sigma(\xi))))$

$\varrho_{\mathrm{l}}((P,\sigma_P,b_P) + (Q,\sigma_Q,b_Q)) = \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(\xi)))) + (\eta_{X+Y}(\mathrm{inr}(\sigma_Q(\xi)))))$

$\varrho_{\mathrm{l}}((P,\sigma_P,b_P)|(Q,\sigma_Q,b_Q)) = \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(\xi))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q(\xi)))))$

$\varrho_{\mathrm{l}}((P,\sigma_P,b_P) \setminus L) = \lambda\,\xi.\,\iota_{X+Y}(\eta_{X+Y}(\mathrm{inr}(\sigma_P(\xi))) \setminus L)$

$\varrho_{\mathrm{l}}((P,\sigma_P,b_P)\,[L]) = \lambda\,\xi.\,\iota_{X+Y}(\eta_{X+Y}(\mathrm{inr}(\sigma_P(\xi)))\,[\varphi])$

$\varrho_{\mathrm{l}}(\text{fix }(P,\sigma_P,b_P)) = \lambda\,\xi.\,\iota_{X+Y}(\text{fix }(\eta_{X+Y}(\mathrm{inr}(\sigma_P(\Box :: \xi)))))$

and

$\varrho_{\mathrm{r}}\colon \Sigma(X \times B(X,Y)) \to \mathcal{P}_{\omega 1}\left(\mathcal{A} \times Y^{(1+X)^*}\right)$

$\varrho_{\mathrm{r}}(\varnothing) = \emptyset$

$\varrho_{\mathrm{r}}(\#m) = \emptyset$

$\varrho_{\mathrm{r}}(\alpha\,{\bullet}\,(P,\sigma,b)) = \{(\alpha, \eta_{X+Y} \circ \mathrm{inr} \circ \sigma)\}$

$$\begin{aligned}\varrho_{\mathrm{r}}((P,\sigma_P,b_P) + (Q,\sigma_Q,b_Q)) = {} & \{(\alpha, \eta_{X+Y} \circ \mathrm{inr} \circ \sigma) \mid (\alpha,\sigma) \in b_P\}\\ & \cup \{(\alpha, \eta_{X+Y} \circ \mathrm{inr} \circ \sigma) \mid (\alpha,\sigma) \in b_Q\}\end{aligned}$$

$$\begin{aligned}\varrho_{\mathrm{r}}((P,\sigma_P,b_P)|(Q,\sigma_Q,b_Q)) = {} & \{(\alpha, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma(\xi))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q(\xi)))))) \mid (\alpha,\sigma) \in b_P\}\\ & \cup \{(\alpha, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(\xi))))|(\eta_{X+Y}(\mathrm{inr}(\sigma(\xi)))))) \mid (\alpha,\sigma) \in b_Q\}\\ & \cup \left\{(\tau, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma(\xi))))|(\eta_{X+Y}(\mathrm{inr}(\sigma'(\xi)))))) \;\middle|\; (\alpha,\sigma) \in b_P, (\beta,\sigma'), \alpha = \overline{\beta}\right\}\end{aligned}$$

$\varrho_{\mathrm{r}}((P,\sigma_P,b) \setminus L) = \{(\alpha, \eta_{X+Y} \circ \mathrm{inr} \circ \sigma) \mid (\alpha,\sigma) \in b, \alpha \notin L\}$

$\varrho_{\mathrm{r}}((P,\sigma,b)\,[\varphi]) = \{(\varphi(\alpha), \eta_{X+Y} \circ \mathrm{inr} \circ \sigma) \mid (\alpha,\sigma) \in b\}$

$\varrho_{\mathrm{r}}(\text{fix }(P,\sigma_P,b)) = \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\mathrm{inr}(\sigma(\Box :: \xi)))) \mid (\alpha,\sigma) \in b\}$

Theorem 3.12. *ϱ is dinatural in X and natural in Y.*

Proof. We will consider the two conditions separately:

ϱ is dinatural in X, which means the following diagram commutes for all $X, Y, Z \in \mathsf{Set}$ and $f\colon X \to Z$:

$$\begin{tikzcd}
 & \Sigma(X, B(X,Y)) \arrow[r, "\varrho_{X,Y}"] & B(X, \Sigma^\star(X+Y)) \arrow[dr, "B(\mathrm{id}_X{,}\Sigma^\star(f+\mathrm{id}_Y))"] & \\
\Sigma(X \times B(Z,Y)) \arrow[ur, "\Sigma(\mathrm{id}_X \times B(f{,}\mathrm{id}_Y))"] \arrow[dr, "\Sigma(f \times B(\mathrm{id}_Z{,}\mathrm{id}_Y))"'] & & & B(X, \Sigma^\star(Z+Y)) \\
 & \Sigma(Z, B(Z,Y)) \arrow[r, "\varrho_{Z,Y}"] & B(Z, \Sigma^\star(Z+Y)) \arrow[ur, "B(f{,}\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))"'] &
\end{tikzcd} \tag{3.8}$$

We show this on elements by case distinction on $x \in \Sigma(X \times B(Z,Y))$:

If $x = \varnothing$, then

$$\begin{aligned}& B(\mathrm{id}_X, \Sigma^\star(f + \mathrm{id}_Y))(\varrho_{X,Y}(\Sigma(\mathrm{id}_X \times B(f, \mathrm{id}_Y))(\varnothing)))\\ = {} & B(\mathrm{id}_X, \Sigma^\star(f + \mathrm{id}_Y))(\varrho_{X,Y}(\varnothing))\\ = {} & B(\mathrm{id}_X, \Sigma^\star(f + \mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}(\varnothing)), \emptyset)\\ = {} & (\Sigma^\star(f + \mathrm{id}_Y) \circ (\lambda\,\xi.\,\iota_{X+Y}(\varnothing)), \emptyset)\\ = {} & (\Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y) \circ (\lambda\,\xi.\,\iota_{X+Y}(\varnothing)) \circ f, \emptyset)\end{aligned}$$

$$
\begin{aligned}
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}(\varnothing)),\emptyset)\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))(\varrho_{Z,Y}(\varnothing))\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))(\varrho_{Z,Y}(\Sigma(f\times B(\mathrm{id}_Z,\mathrm{id}_Y))(\varnothing)))
\end{aligned}
$$

If $x = \#m$**, then** we fix

$$
g \coloneqq \lambda\,(P_1,\dots,P_n).\begin{cases}\iota_{X+Y}(\text{fix }(\eta_{X+Y}(\mathrm{inl}(P)))) & m\le n\wedge P_m=\mathrm{inl}(P)\\ \iota_{X+Y}(m) & \text{otherwise}\end{cases}
$$

and then

$$
\begin{aligned}
&\ B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(\varrho_{X,Y}(\Sigma(\mathrm{id}_X\times B(f,\mathrm{id}_Y))(\#m)))\\
&= B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(\varrho_{X,Y}(\#m))\\
&= B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(g,\emptyset)\\
&= (\Sigma^\star(f+\mathrm{id}_Y)\circ g,\emptyset)\\
&= (\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y)\circ g\circ^*(f),\emptyset)\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))\,(g,\emptyset)\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))(\varrho_{Z,Y}(\#m))\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))(\varrho_{Z,Y}(\Sigma(f\times B(\mathrm{id}_Z,\mathrm{id}_Y))(\#m)))
\end{aligned}
$$

If $x = \alpha\bullet(P,\sigma,b)$**, then**

$$
\begin{aligned}
&\ B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(\varrho_{X,Y}(\Sigma(\mathrm{id}_X\times B(f,\mathrm{id}_Y))(\alpha\bullet(P,\sigma,b))))\\
&= B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(\varrho_{X,Y}(\alpha\bullet(P,\sigma_P\circ^*(f),\{(\beta,\sigma\circ^*(f))\mid(\beta,\sigma)\in b\})))\\
&= B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}(\alpha\bullet(\eta_{X+Y}(\mathrm{inr}(\sigma(x)))))),\{(\alpha,\eta_{X+Y}\circ\mathrm{inr}\circ\sigma)\})\\
&= (\Sigma^\star(f+\mathrm{id}_Y)\circ(\lambda\,\xi.\,\iota_{X+Y}(\alpha\bullet(\eta_{X+Y}(\mathrm{inr}(\sigma(x)))))),\{(\alpha,\eta_{X+Y}\circ\mathrm{inr}\circ\sigma)\})\\
&= (\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y)\circ(\lambda\,\xi.\,\iota_{X+Y}(\alpha\bullet(\eta_{X+Y}(\mathrm{inr}(\sigma(x))))))\circ f,\{(\alpha,\eta_{X+Y}\circ\mathrm{inr}\circ\sigma)\})\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}(\alpha\bullet(\eta_{X+Y}(\mathrm{inr}(\sigma(x)))))),\{(\alpha,\eta_{X+Y}\circ\mathrm{inr}\circ\sigma)\})\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))(\varrho_{Z,Y}(\alpha\bullet(f(P),\sigma,b)))\\
&= B(f,\Sigma^\star(\mathrm{id}_Z+\mathrm{id}_Y))(\varrho_{Z,Y}(\Sigma(f\times B(\mathrm{id}_Z,\mathrm{id}_Y))(\alpha\bullet(P,\sigma,b))))
\end{aligned}
$$

If $x = (P,\sigma_P,b_P)|(Q,\sigma_Q,b_Q)$**, then**

$$
\begin{aligned}
&\ B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(\varrho_{X,Y}(\Sigma(\mathrm{id}_X\times B(f,\mathrm{id}_Y))((P,\sigma_P,b_P)|(Q,\sigma_Q,b_Q))))\\
&= B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))(\varrho_{X,Y}((P,\sigma_P\circ^*(f),\{(\beta,\sigma\circ^*(f))\mid(\beta,\sigma)\in b_P\})\\
&\qquad\qquad\qquad\qquad\qquad\mid(Q,\sigma_Q\circ^*(f),\{(\beta,\sigma\circ^*(f))\mid(\beta,\sigma)\in b_Q\})))\\
&= B(\mathrm{id}_X,\Sigma^\star(f+\mathrm{id}_Y))\\
&\quad((\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q(^*(f)(\xi))))))),\\
&\quad\ (\{(\alpha,\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q(^*(f)(\xi)))))))\mid(\alpha,\sigma)\in b_P\}\\
&\quad\ \ \cup\{(\alpha,\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma(^*(f)(\xi)))))))\mid(\alpha,\sigma)\in b_Q\}\\
&\quad\ \ \cup\big\{(\tau,\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma'(^*(f)(\xi)))))))\,\big|\,(\alpha,\sigma)\in b_P,(\beta,\sigma'),\alpha=\overline{\beta}\big\}))\\
\\
&= (\Sigma^\star(f+\mathrm{id}_Y)\circ(\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q(^*(f)(\xi))))))),\\
&\quad(\{(\alpha,\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q(^*(f)(\xi)))))))\mid(\alpha,\sigma)\in b_P\}\\
&\quad\cup\{(\alpha,\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma(^*(f)(\xi)))))))\mid(\alpha,\sigma)\in b_Q\}\\
&\quad\cup\big\{(\tau,\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma(^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma'(^*(f)(\xi)))))))\,\big|\,(\alpha,\sigma)\in b_P,(\beta,\sigma'),\alpha=\overline{\beta}\big\}))
\end{aligned}
$$

$$\begin{aligned}
&= (\Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y) \circ (\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q({}^*(f)(\xi))))))) \circ f,\\
&\quad (\{(\alpha, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q({}^*(f)(\xi))))))) \mid (\alpha,\sigma) \in b_P\}\\
&\quad \cup \{(\alpha, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\xi))))))) \mid (\alpha,\sigma) \in b_Q\}\\
&\quad \cup \big\{(\tau, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma'({}^*(f)(\xi))))))) \,\big|\, (\alpha,\sigma) \in b_P, (\beta,\sigma'), \alpha = \overline{\beta}\big\}))
\end{aligned}$$

$$\begin{aligned}
&= B(f, \Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q({}^*(f)(\xi))))))),\\
&\quad (\{(\alpha, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma_Q({}^*(f)(\xi))))))) \mid (\alpha,\sigma) \in b_P\}\\
&\quad \cup \{(\alpha, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma_P({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\xi))))))) \mid (\alpha,\sigma) \in b_Q\}\\
&\quad \cup \big\{(\tau, \lambda\,\xi.\,\iota_{X+Y}((\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\xi)))))|(\eta_{X+Y}(\mathrm{inr}(\sigma'({}^*(f)(\xi))))))) \,\big|\, (\alpha,\sigma) \in b_P, (\beta,\sigma'), \alpha = \overline{\beta}\big\}))
\end{aligned}$$

$$\begin{aligned}
&= B(f, \Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y))(\varrho_{Z,Y}((f(P), \sigma_P, b_P)|(f(Q), \sigma_Q, b_Q)))\\
&= B(f, \Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y))(\varrho_{Z,Y}(\Sigma(f \times B(\mathrm{id}_Z, \mathrm{id}_Y))((P, \sigma_P, b_P)|(Q, \sigma_Q, b_Q))))
\end{aligned}$$

The cases for sum, restriction and renaming follow analogously, so let us finally consider the fixpoint operator:

If $x = \mathrm{fix}\ (P, \sigma_P, b)$**, then**

$$\begin{aligned}
&B(\mathrm{id}_X, \Sigma^\star(f + \mathrm{id}_Y))(\varrho_{X,Y}(\Sigma(\mathrm{id}_X \times B(f, \mathrm{id}_Y))(\mathrm{fix}\ (P, \sigma, b))))\\
&= B(\mathrm{id}_X, \Sigma^\star(f + \mathrm{id}_Y))(\varrho_{X,Y}(\mathrm{fix}\ (P, \sigma \circ^* (f), \{(\beta, \sigma \circ^* (f)) \mid (\beta, \sigma_P) \in b\})))\\
&= B(\mathrm{id}_X, \Sigma^\star(f + \mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}(\mathrm{fix}\ (\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))))),\\
&\qquad\qquad \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))) \mid (\alpha, \sigma) \in b\})\\
&= (\Sigma^\star(f + \mathrm{id}_Y) \circ (\lambda\,\xi.\,\iota_{X+Y}(\mathrm{fix}\ (\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))))),\\
&\qquad\qquad \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))) \mid (\alpha, \sigma) \in b\})\\
&= (\Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y) \circ (\lambda\,\xi.\,\iota_{X+Y}(\mathrm{fix}\ (\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))))) \circ f,\\
&\qquad\qquad \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))) \mid (\alpha, \sigma) \in b\})\\
&= B(f, \Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y))((\lambda\,\xi.\,\iota_{X+Y}(\mathrm{fix}\ (\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))))),\\
&\qquad\qquad \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\mathrm{inr}(\sigma({}^*(f)(\Box :: \xi))))) \mid (\alpha, \sigma) \in b\})\\
&= B(f, \Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y))(\varrho_{Z,Y}(\mathrm{fix}\ (f(P), \sigma, b)))\\
&= B(f, \Sigma^\star(\mathrm{id}_Z + \mathrm{id}_Y))(\varrho_{Z,Y}(\Sigma(f \times B(\mathrm{id}_Z, \mathrm{id}_Y))(\mathrm{fix}\ (P, \sigma, b))))
\end{aligned}$$

ϱ **is natural in** Y**,** i.e. the following diagram commutes for all $g\colon Y \to W$:

$$\begin{tikzcd}
\Sigma(X \times B(X,Y)) \arrow[r, "\varrho_{X,Y}"] \arrow[d, "\Sigma(\mathrm{id}_X \times B(\mathrm{id}_X, g))"'] & B(X, \Sigma^\star(X+Y)) \arrow[d, "B(\mathrm{id}_X, \Sigma^\star(\mathrm{id}_X + g))"] \\
\Sigma(X \times B(X,W)) \arrow[r, "\varrho_{X,W}"] & B(X, \Sigma^\star(X+W))
\end{tikzcd} \tag{3.9}$$

Again, we show this on elements by case distinction on $x \in \Sigma(X \times B(X,Y))$:

If $x = \varnothing$**, then**

$$\begin{aligned}
&B(\mathrm{id}_X, \Sigma^\star(\mathrm{id}_X + g))(\varrho_{X,Y}(\varnothing))\\
&= B(\mathrm{id}_X, \Sigma^\star(\mathrm{id}_X + g))((\lambda\,\xi.\,\iota_{X+Y}(\varnothing)), \emptyset)\\
&= ((\Sigma^\star(\mathrm{id}_X + g)(\lambda\,\xi.\,\iota_{X+Y}(\varnothing)) \circ^* (\mathrm{id}_X)), \emptyset)\\
&= ((\lambda\,\xi.\,\iota_{X+Y}(\varnothing)), \emptyset)\\
&= \varrho_{X,W}(\varnothing)\\
&= \varrho_{X,W}(\Sigma(\mathrm{id}_X \times B(\mathrm{id}_X, g))(\varnothing))
\end{aligned}$$

If $x = \#m$, then we fix

$$h := \lambda\,(P_1, \dots, P_n).\begin{cases} \iota_{X+Y}(\text{fix }(\eta_{X+Y}(\text{inl}(P)))) & m \le n \wedge P_m = \text{inl}(P) \\ \iota_{X+Y}(m) & \text{otherwise} \end{cases}$$

and then

$$\begin{aligned} & B(\text{id}_X, \Sigma^\star(\text{id}_X + g))(\varrho_{X,Y}(\#m)) \\ =\ & B(\text{id}_X, \Sigma^\star(\text{id}_X + g))(h, \emptyset) \\ =\ & (\Sigma^\star(\text{id}_X + g) \circ h, \emptyset) \\ =\ & \varrho_{X,W}(\#m) \\ =\ & \varrho_{X,W}(\Sigma(\text{id}_X + B(\text{id}_X, g))(\#m)) \end{aligned}$$

If $x = \alpha\,\bullet\,(P, \sigma, b)$, then

$$\begin{aligned} & B(\text{id}_X, \Sigma^\star(\text{id}_X + g))(\varrho_{X,Y}(\alpha\,\bullet\,(P, \sigma, b))) \\ =\ & B(\text{id}_X, \Sigma^\star(\text{id}_X + g))((\lambda\,\xi.\,\iota_{X+Y}(\alpha\,\bullet\,(\eta_{X+Y}(\text{inr}(\sigma(\xi)))))), \{(\alpha, \eta_{X+Y} \circ \text{inr} \circ \sigma)\}) \\ =\ & (\Sigma^\star(\text{id}_X + g) \circ (\lambda\,\xi.\,\iota_{X+Y}(\alpha\,\bullet\,(\eta_{X+Y}(\text{inr}(\sigma(\xi)))))), \{(\alpha, \Sigma^\star(\text{id}_X + g) \circ \eta_{X+Y} \circ \text{inr} \circ \sigma)\}) \\ =\ & ((\lambda\,\xi.\,\iota_{X+W}(\alpha\,\bullet\,\eta_{X+W}(\text{inr}(g(\sigma(\xi)))))), \{(\alpha, \eta_{X+W} \circ \text{inr} \circ g \circ \sigma)\}) \\ =\ & \varrho_{X,W}(\alpha\,\bullet\,(P, g \circ \sigma, \{(\beta, g \circ \sigma') \mid (\beta, \sigma') \in b\})) \\ =\ & \varrho_{X,W}(\Sigma(\text{id}_X \times B(\text{id}_X, g))(\alpha\,\bullet\,(P, \sigma, b))) \end{aligned}$$

The cases for parallel composition, sum, restriction and renaming follow analogously, so let us finally consider the fixpoint operator:

If $x = \text{fix }(P, \sigma, b)$, then

$$\begin{aligned} & B(\text{id}_X, \Sigma^\star(\text{id}_X + g))(\varrho_{X,Y}(\text{fix }(P, \sigma, b))) \\ =\ & B(\text{id}_X, \Sigma^\star(\text{id}_X + g))((\lambda\,\xi.\,\iota_{X+Y}(\text{fix }(\eta_{X+Y}(\text{inr}(\sigma(\Box :: \xi)))))), \\ & \qquad\qquad \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\text{inr}(\sigma_P(\Box :: \xi)))) \mid (\alpha, \sigma_P) \in b\}) \\ =\ & (\Sigma^\star(\text{id}_X + g) \circ (\lambda\,\xi.\,\iota_{X+Y}(\text{fix }(\eta_{X+Y}(\text{inr}(\sigma(\Box :: \xi)))))), \\ & \quad \{(\alpha, \lambda\,\xi.\,\Sigma^\star(\text{id}_X + g)(\eta_{X+Y}(\text{inr}(\sigma_P(\Box :: \xi))))) \mid (\alpha, \sigma_P) \in b\}) \\ =\ & ((\lambda\,\xi.\,\iota_{X+Y}(\text{fix }(\eta_{X+W}(\text{inr}(g(\sigma(\Box :: \xi))))))), \\ & \quad \{(\alpha, \lambda\,\xi.\,\eta_{X+Y}(\text{inr}(\sigma_P(\Box :: \xi)))) \mid (\alpha, \sigma_P) \in \{(\beta, g \circ \sigma') \mid (\beta, \sigma') \in b\}\}) \\ =\ & \varrho_{X,W}(\text{fix }(P, g \circ \sigma, \{(\beta, g \circ \sigma') \mid (\beta, \sigma') \in b\})) \\ =\ & \varrho_{X,W}(\Sigma(\text{id}_X \times B(\text{id}_X, g))(\text{fix }(P, \sigma, b))) \end{aligned}$$

□

Now that we have defined ϱ, we plug it into the framework of HO-GSOS, as discussed in Section 2.3.2. That means that we obtain a $B(\mu\Sigma, -)$-coalgebra $\gamma\colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$. Taking the pullback of **coit** γ with itself we get strong bisimulation on guarded CCS processes with fixpoints as a Σ-congruence.

4 Notes on the Agda-Formalization

Much of this thesis has been formalized in the proof assistant Agda [Agd]. The implementation can be found at https://github.com/flogth/ccs. This section will highlight some of the choices made and difficulties encountered.

Representation of the Syntax The formalization takes an approach to representing the syntax of CCS that is slightly different from the one presented in this thesis: Instead of having a fixed type Proc of terms, we define a family $(\text{Proc } n)_{n \in \mathbb{N}}$ of types where $P \colon \text{Proc } n$ is a term with at most n free variables. This idea was adapted from Programming Language Foundations in Agda (PLFA) [WKS22], were it is used to implement the simply-typed λ-calculus. While they index their family of terms over lists of types (contexts), the untyped nature of CCS allows us to simplify the index to just natural numbers. Sticking to this representation of terms allows us to easily adapt large parts of the meta theory regarding substitutions from PLFA.

The two constructors of the Proc family where this becomes relevant are the ones for variables and fixpoints. The variable constructor `#_` ensures that we cannot refer to any variable greater than n:

```
#_ : ∀ {n} → (x : Fin n) → Proc n
```

The type `Fin n` is given in the standard library [The24] and corresponds to the set $\{0, 1, \ldots, n-1\}$.

For the fixpoint operator, we need to ensure that the body of the term is allowed to reference exactly one more variable than the whole term, namely the variable being bound:

```
fix : ∀ {n} → (P : Proc (suc n)) → Proc n
```

Some of the meta theory around substitutions, first and foremost the `subst-commute` theorem are again adapted from PLFA. That formalization relies heavily on the axiom of functional extensionality, which is not given by the type theory of Agda and has to be assumed separately. Fortunately, one can work around this by simply restructuring the proof such that no propositional equality between functions is needed. Instead, the pointwise equality defined in the standard library suffices.

Definition of ϱ In order to implement the map ϱ in our formalization, we first need to define a few categorical preliminaries that are not present by default. For instance, we need to give a signature functor Σ as in Equation (3.6). We also need to give the free algebra $\Sigma^\star X$ for any type X. Fortunately, we can adapt this from Goncharov and Vatthauer [GV25].

Since the type of terms is now a family, we need to give a separate type for the initial algebra of Σ:

$$\coprod_{n \in \mathbb{N}} \text{Proc } n$$

In Agda, this corresponds to the type

```
μΣ : Set ℓ
μΣ = Σ ℕ λ n → Proc n
```

This slightly complicates some inductive definitions, since the Agda termination checker cannot infer when a definition is guarded on the second projection of type Proc n. To be able to state these definitions, one first has to “curry” the dependent pair, so the termination checker can validate the arguments separately.

5 Summary

We have presented how to encode the operational semantics of guarded CCS in the framework of higher-order GSOS. This involved finding an operational semantics that fits the HO-GSOS rule format and proving its equivalence to the standard operational semantics. This was done by defining several intermediate operational semantics to construct a chain of equivalences. Here is also the point where the guardedness assumption comes into play since the equivalence only holds for guarded terms.

The operational semantics was then encoded as a dinatural transformation that captures the operational rules. By the framework of HO-GSOS we obtained a coalgebraic structure on the initial algebra that allowed us to define strong bisimulation on CCS processes. This notion of strong bisimulation is a Σ-congruence, a result which we obtain for free by compositionality of our semantics.

The work presented here has focused on strong bisimilarity, since that is provided by the HO-GSOS framework. Introducing a recursion operator in the straightforward way would have forced us to introduce silent transititions, which is in conflict with strong bisimilarity. In later work, a treatment of weak bisimilarity has been provided [Urb+23]. We conjecture that this can be used to implement the fixpoint operator of CCS in a more straightforward way, at least with respect to weak bisimulation.

Bibliography

[AC98] Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[Agd] Agda Developers. *Agda.* Version 2.8.0. URL: https://agda.readthedocs.io/.

[AP86] K. R. Apt and G. D. Plotkin. “Countable nondeterminism and random assignment”. In: *J. ACM* 33.4 (Aug. 1986), pp. 724–767. ISSN: 0004-5411. DOI: 10.1145/6490.6494. URL: https://doi.org/10.1145/6490.6494.

[Bar70] Michael Barr. “Coequalizers and free triples”. In: *Mathematische Zeitschrift* 116 (1970), pp. 307–322.

[BIM95] Bard Bloom, Sorin Istrail and Albert R. Meyer. “Bisimulation can’t be traced”. In: *J. ACM* 42.1 (Jan. 1995), pp. 232–268. ISSN: 0004-5411. DOI: 10.1145/200836.200876. URL: https://doi.org/10.1145/200836.200876.

[BK82] Jan A. Bergstra and Jan Willem Klop. *Fixed point semantics in process algebras.* Tech. rep. 1982.

[de 72] N.G de Bruijn. “Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem”. In: *Indagationes Mathematicae (Proceedings)* 75.5 (1972), pp. 381–392. ISSN: 1385-7258. DOI: https://doi.org/10.1016/1385-7258(72)90034-0.

[Gon+24] Sergey Goncharov et al. “Higher-Order Mathematical Operational Semantics”. In: *CoRR* (2024).

[GV25] Sergey Goncharov and Leon Vatthauer. *Agda Meta Semantics.* Jan. 2025. URL: https://github.com/sergey-goncharov/agda-meta-semantics.

[Hoa78] C. A. R. Hoare. “Communicating sequential processes”. In: *Commun. ACM* 21.8 (Aug. 1978), pp. 666–677. ISSN: 0001-0782. DOI: 10.1145/359576.359585. URL: https://doi.org/10.1145/359576.359585.

[JT98] Achim Jung and Regina Tix. “The Troublesome Probabilistic Powerdomain”. In: *Electronic Notes in Theoretical Computer Science* 13 (1998). Comprox III, Third Workshop on Computation and Approximation, pp. 70–91. ISSN: 1571-0661. DOI: https://doi.org/10.1016/S1571-0661(05)80216-6.

[Mil80] Robin Milner. *A Calculus of Communicating Systems.* 1980. DOI: https://doi.org/10.1007/3-540-10235-3.

[Mil83] Robin Milner. “Calculi for synchrony and asynchrony”. In: *Theoretical Computer Science* 25.3 (1983), pp. 267–310. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(83)90114-7.

[Mil89] Robin Milner. *Communication and concurrency.* USA: Prentice-Hall, Inc., 1989. ISBN: 0131150073.

[Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus.* Cambridge University Press, 1999. ISBN: 978-0-521-65869-0.

[Plo04] Gordon D. Plotkin. “A structural approach to operational semantics”. In: *J. Log. Algebraic Methods Program.* 60-61 (2004), pp. 17–139.

[The24] The Agda Community. *Agda Standard Library*. Version 2.1.1. Sept. 2024. URL: https://github.com/agda/agda-stdlib.

[TP97] D. Turi and G. Plotkin. “Towards a mathematical operational semantics”. In: *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*. 1997, pp. 280–291. DOI: 10.1109/LICS.1997.614955.

[Urb+23] Henning Urbat et al. *Weak Similarity in Higher-Order Mathematical Operational Semantics*. 2023. arXiv: 2302.08200 [cs.PL].

[WKS22] Philip Wadler, Wen Kokke and Jeremy G. Siek. *Programming Language Foundations in Agda*. Aug. 2022. URL: https://plfa.inf.ed.ac.uk/22.08/.