Radboud University Nijmegen

Faculty of Science

Comparing Formats for Name-Passing Process Calculi

Thesis MSc Mathematics

Supervisor:
Dr. J. C. (Jurriaan) Rot

Author:
Erik Voogd

Second reader:
Dr. T. T. (Thorsten) Wißmann

Abstract

We study rule specification formats for transition systems of nominal sets that support name binding and name substitutions. Rule specifications of name-passing process calculi should induce transition systems where binding names in transition labels can be subject to alpha-conversion. Name binding in a process is modeled in the syntax by name abstraction in algebras of nominal sets. In a similar way, we show that name binding in transition labels can be modeled by name abstraction in coalgebras of nominal sets under certain conditions.

August 31, 2021

Contents

0 Introduction

Today's world is full of software and communication systems in which processes interact with each other throughout. For our everyday life, it is crucial that such processes run safely and that they do not exhibit unexpected behaviour. In order to define and reason with such philosophical concepts as safety and well-behavedness, communication systems are cast into the mathematical framework of *labeled transition systems* (LTS), in which steps of computation or communication are mathematically formalized and made explicit.

A *process calculus* constitutes a popular way to specify labeled transition systems for communication and concurrent computation. Indeed, states of an LTS for a process calculus are interpreted as processes running in parallel and passing messages. A process calculus can be viewed as a high-level programming language: it consists of a *syntax*, specifying how the structure of a process is written, along with a *semantics*, defining execution for processes by describing the way syntax can transform over time. Two prominent examples of process calculi are CCS [13] and the π-calculus [14, 21], the latter being an extension of the former by also allowing communication channels themselves to be passed around.

The syntax of a calculus is given by providing a context-free grammar, and a semantics is then given by providing a collection of rules that specify appropriate transitions from states to other states. The rules for transitions in a process calculus specify the behaviour of the LTS, and different contexts ask for different specifications. Indeed, many different kinds of specifications can be formulated, and many different kinds of process calculi exist [17]. Even 'the' π-calculus should rather be thought of as a *family* of calculi, since many different variants exist by adding, omitting, or varying certain rules of syntax and behaviour.

In the present thesis we are interested in a uniform look at different process calculi. Hence, we consider *formats* to which rules of a specification can (or should) adhere, meaning that the rules should be of a certain shape. In doing so, we study a whole class of process calculi can be specified by rules in a given format. Taking this approach, we start to see some important intrinsic properties of the class of calculi that are allowed by the formats. The *GSOS rule format* by Bloom et al. [4], for example, ensures *congruence of bisimilarity*. Although often tedious to prove, this property is much sought after, as it enables us to reason about processes in a compositional way.

The GSOS rule format was originally developed within the context of CCS and similar calculi, such as CSP. Staton and Fiore [7] have proposed a new rule format that accounts for name binding (locally instantiating new names in a process) and name substitution, as required by many modern name-passing process calculi, such as the π-calculus. Their format can be considered quite restrictive, but it does ensure congruence of bisimilarity.

A different format for name-passing process calculi, proposed by Aceto et al. [1], ensures that binding names in transition labels can be locally renamed to "suitably fresh" names. That is, we can rename the bound names in the label with names that do not already occur locally. The format is less restrictive than the GSOS-like format by Staton and Fiore [7], but it does not guarantee congruence of bisimilarity.

The purpose of this thesis is to discuss the different intents and outcomes, advantages and disadvantages regarding the two formats we discussed above, and how they relate. Most of the theory in this thesis is well established in the literature. Section 4, however, contains results that are used in [7], but have not been made explicit or proven. There, we show that, under certain assumptions, we can model name binding in transition system labels using name abstraction in the corresponding coalgebra in the category of nominal sets. We then discuss how this relates to the notion of *alpha-conversion of residuals* proposed in [6, 24].

The structure of this thesis is as follows: in Section 1, we familiarize the reader in an informal way with some basic notions such as transition systems and their specifications, with the importance of congruence of bisimilarity, and with name binding and substitution of names. After that, in Section 2, we formally introduce transition systems and their foundation in category theory. Readers not familiar with category theory can turn to Appendix A for an introduction of the central categorical notions for this thesis. In Section 3, we develop the theory of nominal sets, which is useful for the notions of name bindings, freshness, and name substitutions. In Section 4, we discuss transition systems of nominal sets, and discuss some properties they should pursue. Finally, in Section 5 we make some concluding remarks about the two different formats and how they relate, and discuss possible further lines of research.

1 Problem Statement

In this section we familiarize the reader with transition systems and process calculi. We also explain why congruence of bisimilarity is an important property for process calculi, and we indicate some complications that arise when working with name binding, and name substitution. Finally, we reiterate the intention of this thesis.

1.1 Models of concurrency and communication

The leading example of a process calculus for modeling concurrent computation and communication is the *Calculus of Communicating Systems* (CCS) [13]. In CCS, a process modeled as a black box that interacts with the outside world by performing synchronized communication with other black boxes.

Consider, for example, a typical interaction with a coffee machine at the office. The interaction involves two autonomous actors: the person and the machine, both referred to as *processes*. From the viewpoint of the person, a coin is given to the machine, a choice between coffee or tea is made by pressing a button req-cof or req-tea, and then the machine gives you the drink accordingly. This process from the person's perspective can be graphically represented as

$$\begin{tikzcd}
\boxed{P_1} \arrow[r, "\overline{\mathsf{coin}}"] & \boxed{P_2} \arrow[r, "\overline{\mathsf{req\text{-}cof}}"] & \boxed{P_3} \arrow[r, "\mathsf{coffee}"] & \boxed{P_4}
\end{tikzcd}$$

Here, the boxes represent *states* in which the person may find itself, and the arrows indicate transitions to subsequent states. For instance, a process (person) in state P_1 is ready to *output* (indicated by the overlining) a coin, and transition to state P_2, which, in turn, is ready to output the choice of requesting a cup of coffee. Finally, P_3 is ready to receive coffee, after which it terminates (one ought to treat the analogy of the person as a process loosely when talking about termination).

The dot '.' is often used to represent sequential actions, and so the process P_1 is written down as

$$P_1 \triangleq \overline{\mathsf{coin}} \,.\, \overline{\mathsf{req\text{-}cof}} \,.\, \mathsf{coffee} \,.\, \mathsf{nil}$$

where nil is a terminated process. Here, nil is preceded by three action prefixes.

Communication can only happen if there are two or more parties involved, so let us imagine what should happen at the side of the coffee machine:

$$\begin{tikzcd}
 & & \boxed{M_3} \arrow[dll, bend right, "\overline{\mathsf{coffee}}"'] \\
\boxed{M_1} \arrow[r, "\mathsf{coin}"] & \boxed{M_2} \arrow[ur, "\mathsf{req\text{-}cof}"] \arrow[dr, "\mathsf{req\text{-}tea}"'] & \\
 & & \boxed{M_4} \arrow[ull, bend left, "\overline{\mathsf{tea}}"]
\end{tikzcd}$$

There are no terminated processes here: every state is ready to do something. This reflects the fact that a coffee machine should always be operational, even after having served one person. The state of M_2 is even ready to accept two inputs: *either* a request for coffee *or* a request for tea. This choice is nondeterministic from the viewpoint of the machine. We get a recursive definition that also involves a '+' for this nondeterministic choice:

$$M_1 \triangleq \text{coin} . (\text{req-cof} . \overline{\text{coffee}} . M_1 + \text{req-tea} . \overline{\text{tea}} . M_1)$$

Since P_1 is ready to output a coin ($P_1 \xrightarrow{\overline{\text{coin}}} P_2$), and M_1 can input a coin ($M_1 \xrightarrow{\text{coin}} M_2$), the two can interact if they are executed in parallel: $P_1 \parallel M_1 \xrightarrow{\tau} P_2 \parallel M_2$. The '$\parallel$' here denotes parallel composition, signifying that we run both operands concurrently. The label τ indicates that this transition happens *silently*, meaning it involves no specific names or labels, as the two processes are synchronized on a shared channel. (It might have been more clear if there actually *was* no name or label, but we give the silent transition a special label in order to uphold the name *labeled* transition system...)

Ideally, we want a calculus such as CCS to allow us to model many, possibly infinite different kinds of processes, and in practice, they do. For each one of these infinite processes, then, how do we know what transitions are allowed for it? Even though the number of processes itself is infinite, the way we write processes is defined by a finite amount of rules. We saw, as an example for a syntax rule, that if P and Q are processes, then $P \parallel Q$ is also a process.

For all the syntax rewrite rules that were used to obtain the syntax of a process, there was one rule used last to write it down. By providing a collection of behavioural rules that apply for the syntax rule that was used last, we can specify what transitions are appropriate in the system. A common example of such a rule is the rule for synchronization that we saw in the example of the coffee machine:

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \tag{1}$$

Written like this, using any P, P', Q, Q' and α, if the transitions above the line (the *premises*) are valid, then the transition below the line (the *conclusion*) is also valid. This way, a rule specification can inductively define a transition system. Not all formats induce well-defined transition systems, as we can formulate paradoxical or contradictory rules.

1.2 Processes that are equal

So this is how we can mathematically formalize communication protocols of getting coffee from a machine in a labeled transition system. In order to reason about topics such as safety and well-behavedness of more complex systems, it is important to talk about equality of such systems. How should we define an equality of processes?

It must be said that equality of processes should be concerned more with execution than with syntax. In $\begin{tikzcd} \boxed{P} \arrow[r, "\alpha"] & \boxed{Q} \arrow[loop right, "\alpha"] \end{tikzcd}$, the states P and Q seem to be distinct. However, both processes can only perform the sequence of actions $\alpha, \alpha, \alpha, \ldots$ ad infinitum, which are called *traces*. When we look at their behaviour, therefore, they seem equal. This kind of equality of processes in an operational sense is called *trace equivalence* of processes.

Back to our coffee machine. Notice that, for each time a person interacts with the coffee machine, the state M_1 has two possible traces, namely $M_1 \xrightarrow{\text{coin}} M_2 \xrightarrow{\text{req-cof}} M_3 \xrightarrow{\overline{\text{coffee}}}$

M_1 and $M_1 \xrightarrow{\text{coin}} M_2 \xrightarrow{\text{req-tea}} M_3 \xrightarrow{\overline{\text{tea}}} M_1$. But if we were to prepone the decision for either coffee or tea to the moment of inserting a coin –a poor choice of coffee machine design, but alas–, we get the following labeled transition system:

$$\begin{tikzcd}
 & \boxed{M_2'} \arrow[r, "\text{req-cof}"] & \boxed{M_3'} \arrow[dll, bend right, "\overline{\text{coffee}}"'] \\
\boxed{M_1'} \arrow[ur, "\text{coin}"] \arrow[dr, "\text{coin}"'] & & \\
 & \boxed{M_4'} \arrow[r, "\text{req-tea}"'] & \boxed{M_5'} \arrow[ull, bend left, "\overline{\text{tea}}"]
\end{tikzcd}$$

Notice that M_1 and M_1' are trace equivalent (write $M_1 \sim M_1'$). So, do we want to say that M_1 and M_1' are equal as processes?

Arguably not: a person that needs her daily dosage of caffeine might run into trouble when using the coffee machine represented by M_1'. Indeed, applying a common rule for synchronization (1), $P_1 \parallel M_1' \xrightarrow{\tau} P_2 \parallel M_4'$ is a valid transition. But M_4' can only accept a request for tea, whereas P_2 absolutely needs coffee. It would seem that both parties are at an impasse and cannot communicate further. This unfortunate situation cannot occur with our first model of the machine, represented by M_1. Although the processes on their own are able to execute the same things, when put in the context of parallel composition with P_1, different possible behaviours arise. This means that trace equivalence need not be a *congruence*, and is therefore a questionable choice for equality. Congruence means precisely that if M_1 is 'equal' to M_1', then, if we were to put M_1 in a certain context, it will be 'equal' to M_1' put in the same context. The context here can really mean anything involving some operation: not only parallel composition, but also nondeterministic choice ($P + Q$) and action prefixing ($\alpha \, . \, P$). This property of congruence for equality is only natural when we consider congruence of equality of numbers, e.g. $1 + 1 = 2$, so therefore $3 \cdot (1 + 1) = 3 \cdot 2$.

Instead of trace equivalence, *bisimilarity* is often used to equate processes. Two bisimilar states P and Q always satisfy the following property: if P can make a labeled transition to P', then Q can make a labeled transition (with the same label) to some process Q' such that P' and Q' are again bisimilar. The process M_2 can accept requests for coffee and tea, but M_4' can only accept requests for tea, so M_2 and M_4' cannot be bisimilar. But then, by the property we just described, also M_1 and M_2' cannot be bisimilar. So, could bisimilarity be a good relation to use for equality of processes? That is, do we have *congruence of bisimilarity*?

To reason about processes in a compositional manner, congruence of bisimilarity is a paramount property for labeled transition systems. However, proofs that the property holds often are long and tedious. To avoid such proofs, Bloom et al. [4] invented the *GSOS format* for rules of behaviour such as (1). A *rule format* restricts rules to a certain shape, with the purpose of obtaining desirable properties. The GSOS rule format is one of many formats [16, 2], and ensures that the specification induces a well-defined transition system that moreover enjoys congruence of bisimilarity. This property comes free of proof, since the format already guarantees it!

In further research, Turi and Plotkin [23] formalized specifications in the GSOS format as *abstract rules* and showed how they can be seen as rigid mathematical objects (distributive laws) defined in the language of category theory. This way, CCS can be seen as a distributive law between endofunctors on the category of sets.

1.3 Receiving data

There are some limitations of CCS. Our coffee machine, for example, can only accept requests for tea and coffee. Communications involving arbitrary data can easily be imagined (e.g., an amount of money, a number for a menu option, etc.), but CCS, in its original design, does not support the transfer of data, only synchronization of names such as coin.

In contrast, the π-calculus [14, 21] is a process calculus that does support transfer of data! In the π-calculus, we could write a person (process) that outputs the choice for cappuccino through a channel x as $P \triangleq x\langle \mathsf{cappu} \rangle \,.\, P'$ (angled brackets signify output) and a coffee machine accepting such a choice through the same channel x as $M \triangleq x(y) \,.\, M'$ (round brackets signify input). Here, x is the channel through which person and machine communicate, y is the name of an input variable for the machine, and P' and M' are processes that describe further communication between the two parties.

In the π-calculus, processes can synchronize too, but there is an additional element to it. The two transitions $P \xrightarrow{x\langle \mathsf{cappu} \rangle} P'$ and $M \xrightarrow{x(y)} M'$ together become

$$P \parallel M \xrightarrow{\tau} P' \parallel M'\{\mathsf{cappu}/y\} \tag{2}$$

The extra element here, indicated by the curly brackets, is that every occurrence of the variable y in M' should be *substituted* with the name cappu after the synchronization is performed. This way, our model makes sure that the coffee machine actually receives the choice of the person!

1.4 Bound to be free

The action prefix $x(y)$ in the process $x(y) \,.\, M'$ above instantiates a new piece of data y for M'. The variable y here is an example of a *binding name*: the name occurs bound within the scope of M'. Other binding constructs besides input exist. For example, in the π-calculus, a term $(\boldsymbol{\nu} x)P$ creates a new name that can be used within the scope of the process P, much like $\lambda x \,.\, M$ in the λ-calculus creates a new name x with scope M.

A substitution such as $\{\mathsf{cappu}/y\}$ is required to be *capture-avoiding*: it should leave all binding names in M' unaffected, even though we may locally also write y for them. A capture-avoiding substitution hence only affects *free* names, i.e., names that do not occur under some binding, such as input. It is often written that when names of such a substitution do coincide, then the binding name can always be *'silently renamed'*. Whether a name is called *free* or *bound*, depends on the context: in $M \triangleq x(y) \,.\, M'$, for instance, y occurs bound, but in M' (without the input action prefix after a transition $M \xrightarrow{x(y)} M'$), it occurs free.

For a binding name, this 'silent renaming' means that one can actually choose to write down any name, as long as it does not conflict with other, already existing free names within the same scope. A process $x(y)\ .\ x\langle y\rangle\ .\ \mathsf{nil}$, for example, can and should be considered equivalent to $x(a)\ .\ x\langle a\rangle\ .\ \mathsf{nil}$, since they both represent a process where a free channel x just immediately sends back whatever it receives. Why should it matter what name we give to what it receives? This notion of equivalence through renaming bound names is referred to as *α-equivalence* (denoted $\approx_\alpha$). The formal mathematical construction that models binding names through α-equivalence is called *abstraction* of names.

Example 1.1. Let $Q \triangleq (x(y)\ .\ x\langle y\rangle\ .\ \mathsf{nil}) \parallel (z\langle y\rangle\ .\ \mathsf{nil})$, which has two parallel components, the first of which has a binding name y, instantiated upon input through x. The second parallel component has a free occurrence of y. We have

$$Q\{a/y\} = (x(y)\ .\ x\langle y\rangle\ .\ \mathsf{nil}) \parallel (z\langle a\rangle\ .\ \mathsf{nil}),$$

where the free occurences of y have been substituted, but the bound ones have not. However, note also that $Q \approx_\alpha (x(a)\ .\ x\langle a\rangle\ .\ \mathsf{nil}) \parallel (z\langle y\rangle\ .\ \mathsf{nil})$, since, by α-equivalence, we can rename the binding name y however we like (except as x).

When writing $M = x(y)\ .\ M'$, α-equivalence says that *any* choice of name for y will do (renaming y in M' accordingly). Any name? No, it has to be suitably *fresh*, meaning it should not already occur in the relevant context. We want to rule out the name x, for instance, because it is already the (free) name of the channel.

The notion of α-equivalence also extends to names in transition labels: consider two π-calculus processes

$$P_1 \triangleq x(a)\ .\ x\langle a\rangle\ .\ \mathsf{nil} \qquad \text{and} \qquad P_2 \triangleq x(a)\ .\ x\langle a\rangle\ .\ x\langle d\rangle\ .\ \mathsf{nil},$$

both with one binding name a. Note that P_2 has a free name d that P_1 does not have. Since P_1 is α-equivalent to $x(d)\ .\ x\langle d\rangle\ .\ \mathsf{nil}$, we have that $P_1 \xrightarrow{x(d)} x\langle d\rangle\ .\ \mathsf{nil}$ is a valid transition. However, the situation is different for P_2: the transitions

$$P_2 \xrightarrow{x(a)} x\langle a\rangle\ .\ x\langle d\rangle\ .\ \mathsf{nil} \qquad \text{and} \qquad P_2 \xrightarrow{x(d)} x\langle d\rangle\ .\ x\langle d\rangle\ .\ \mathsf{nil}$$

lead to two distinct processes. Ideally, the second transition should be disallowed. In other words, the binding name a in the label should be allowed to be renamed only to suitably fresh names. This means exactly that the name in the label should be a binding name in the target. This is formalized as *alpha-conversion of residuals* by Aceto et al. [1], a *residual* being the pair consisting of the label and the target of a transition.

1.5 Contribution and related work

The π-calculus has the advantage over CCS that it allows us to pass around names through channels between processes. However, the GSOS rule format is not adequate for name passing process calculi [4], since

(i) the processes in the calculus need to be defined up to α-equivalence,

(ii) rules for behaviour often have side-conditions for freshness of names, and

(iii) bound data in the label should be suitably fresh.

These problems have not been accounted for in the original GSOS rule format and in the category theoretical approach by Turi and Plotkin [23], which used distributive laws between endofunctors on the category of sets.

In this thesis, we focus on the problem mentioned in (iii) and compare the two formats in [1] and [7] that tackle this problem. The latter is a variant of the GSOS rule format, appropriately adjusted for name binding and substitution. This format was developed to ensure that the specification induces a name-passing transition system that satisfies congruence of bisimilarity. The authors construct an abstract rule from a transition system specification in a general setting, and show that this is a distributive law under the condition that the rules are in their format. The coalgebraic structure for the initial bialgebra for this distributive law is now the labeled transition system of interest, and it enjoys congruence of bisimilarity.

In [1], the authors formulate a property called *alpha-conversion of residuals* [6, 24]: any bound name in a label can be renamed to another name that is "suitably fresh". If, for example, $P \xrightarrow{a(b)} P'$ is a valid transition with b a bound name, then $P \xrightarrow{a(c)} P'\{c/b\}$ is also a valid transition for every c that is fresh in P'. A transition system satisfying this property is called a *nominal transition system* (NTS) in their work.

In this thesis, we argue that the interpretation of "suitably fresh" may better also include freshness for the *source* of a transition, rather than just for the *target* and the *label* of the transition. Indeed, Staton & Fiore require binding names to be globally fresh, and they claim that this corresponds exactly to coalgebraic structures using abstraction of names. We elaborate on this claim and discuss its relation to nominal transition systems [1].

Transition systems that support name binding and substitution are established by using the theory of *nominal sets* [18]. Nominal sets are structures that allow us to formalize α-equivalence, freshness, and name substitution. Formats rely heavily on this theory of nominal sets, and so will the content of this thesis.

2 Distributive Laws

In this section we look at transition systems and how they arise concretely from their specifications. In a more abstract context of category theory, transition systems are coalgebras (Section 2.1), their syntax can be described by algebras (Section 2.2), and combined they form bialgebras. Transition system specifications (Section 2.3) can be seen as distributive laws (Section 2.4) between endofunctors describing syntax and behaviour. Bialgebras for distributive laws satisfy congruence of bisimilarity (Section 2.5), and the coalgebra that is part of the initial bialgebra for a distributive law is the transition system of interest (Section 2.6). All of this is well-documented theory, see for example [3, 10].

We need basic notions from category theory to talk about (co)algebras. A reader not familiar with this is strongly advised to turn to Appendix A.

2.1 Coalgebras for System Behaviour

Lists and trees are well-understood examples of *inductive types*, used for storing and manipulating finite amounts of data in computer programs. There are many situations, however, where we require data to carry an infinite structure. For example, certain verification techniques involve automata that accept infinite *streams*. Streams are *coinductive types* that are to be thought of as unending lists. Stream systems are the mathematical models that we use to study these data structures.

Definition 2.1. A *labeled stream system* is a triple $(X, \mathcal{L}, \delta)$, where X is a set of states, $\mathcal{L}$ is a set of labels, and $\delta : X \longrightarrow \mathcal{L} \times X$ is the transition function. ■

If $\delta(x) = (a, y)$, then we say that x can make a transition to y with label $a \in \mathcal{L}$. It is customary, and often convenient, to use arrow notation for transitions. That is, if $\delta(x) = (a, y)$ we may omit the transition function δ by writing a *stream literal* $x \xrightarrow{a} y$, where x and y are the *source* and the *conclusion* of the transition, respectively. Note the graphical distinction between $\longrightarrow$ (for morphisms such as functions and functors) and $\rightarrow$ (for stream transitions).

Example 2.2. Let $X = \{u, v, w\}$ and $\mathcal{L} = \{a, b\}$. Define $\delta : X \longrightarrow \mathcal{L} \times X$ by $\delta(u) = (a, u)$, $\delta(v) = (a, w)$, and $\delta(w) = (b, v)$. There are three transitions in this system, written as the literals $u \xrightarrow{a} u$, $v \xrightarrow{a} w$, and $w \xrightarrow{b} v$:

$$\begin{tikzcd} u \arrow[loop above, "a"] & v \arrow[r, bend left, "a"] & w \arrow[l, bend left, "b"] \end{tikzcd}$$

Here, u is a state that produces the constant stream $(aa\ldots)$, repeating the same label a ad infinitum. The states v and w produce the *alternating* streams $(abab\ldots)$ and $(baba\ldots)$.

The behaviour of stream systems can be described coalgebraically.

Let $B : \mathbb{C} \longrightarrow \mathbb{C}$ be an endofunctor on a category $\mathbb{C}$.

Definition 2.3. A *B-coalgebra* is a pair (X,β), where X is called the carrier, which is an object of $\mathbb{C}$, and $\beta_X : X \longrightarrow BX$ is a morphism of $\mathbb{C}$ called the B-coalgebra structure. ■

We will use notations $\beta : X \longrightarrow BX$, $X \xrightarrow{\beta} BX$, and (X,β) interchangeably to refer to B-coalgebras, or sometimes even just β if X and B are clear from the context.

Example 2.4. In Example 2.2, the pair (X,δ) with $X = \{u,w,v\}$ is an L-coalgebra. Here, $L : \mathbf{Set} \longrightarrow \mathbf{Set}$ is the functor $LY = \mathcal{L} \times Y$ with $\mathcal{L} = \{a,b\}$.

In fact, any labeled stream system $(X,\mathcal{L},\delta)$ as in Definition 2.1 is an L-coalgebra for the functor $L = \mathcal{L} \times (-)$. The set X is the carrier, and the coalgebra structure is just the transition function δ.

For a given endofunctor B, the collection of B-coalgebras forms a category, denoted B-**Coalg**. A morphism between B-coalgebras β and γ is a morphism $f : X \longrightarrow Y$ in $\mathbb{C}$ between the carriers X, Y that respects the coalgebra structure. That is, it makes the following diagram commute:

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\beta"] & Y \arrow[d, "\gamma"] \\ BX \arrow[r, "Bf"] & BY \end{tikzcd}$$

There is an obvious forgetful functor U^B from B-**Coalg** to its underlying category $\mathbb{C}$ (giving the carrier set).

Final or initial B-coalgebras are final or initial objects of B-**Coalg** respectively. The unique morphism from a B-coalgebra to the final B-coalgebra (if it exists) is called the *coinductive extension*.

The following lemma will be useful later:

Lemma 2.5. Let $B : \mathbb{C} \longrightarrow \mathbb{C}$ be an endofunctor on $\mathbb{C}$. An initial object A of $\mathbb{C}$ yields a unique coalgebra structure $a : A \longrightarrow BA$ which is an initial B-coalgebra. ■

Proof. By initiality of A we get a unique coalgebra structure $a : A \longrightarrow BA$. This coalgebra structure is initial: for any $\beta : X \longrightarrow BX$, there is a unique arrow $f : A \longrightarrow X$ by initiality of A. This is a coalgebra morphism, since $Bf \circ a = \beta \circ f : A \longrightarrow BX$, again, by initiality of A. ☺

Example 2.6. Let $\mathcal{L}$ be a set of labels and $LX = \mathcal{L} \times X$ be an endofunctor on **Set**. Recall that $\mathcal{L}^\omega$ denotes the set of streams with elements from $\mathcal{L}$. Then $\mathcal{L}^\omega$ carries a final L-coalgebra structure z given by $z : (l_0 l_1 l_2 \ldots) \mapsto (l_0, (l_1 l_2 \ldots))$. The first component $\mathcal{L}^\omega \longrightarrow \mathcal{L}$ is often called hd (the *head* of the stream) and the second component $\mathcal{L}^\omega \longrightarrow \mathcal{L}^\omega$ is called tl (the *tail* of the stream).

To see that this structure is final, consider an arbitrary L-coalgebra (X, β). We have to define a (unique) function $f_\beta : X \longrightarrow \mathcal{L}^\omega$ such that the diagram

$$\begin{tikzcd} X \arrow[r, dashed, "!f_\beta"] \arrow[d, "(\beta_0,\beta_1)=\beta"'] & \mathcal{L}^\omega \arrow[d, "z=(\mathsf{hd},\mathsf{tl})"] \\ \mathcal{L} \times X \arrow[r, dashed, "\mathrm{id}_\mathcal{L} \times f_\beta"] & \mathcal{L} \times \mathcal{L}^\omega \end{tikzcd}$$

commutes. Observe that we must have $(\mathsf{hd} \circ f_\beta)(x) = (\mathrm{id}_\mathcal{L} \circ \beta_0)(x)$. In words, the first element $l_0 = (\mathsf{hd} \circ f_\beta)(x)$ of the stream $f_\beta(x)$ is $\beta_0(x)$.
For the second and third element we have

$$\begin{aligned} l_1 &= (\mathsf{hd} \circ \mathsf{tl} \circ f_\beta)(x) & l_2 &= (\mathsf{hd} \circ \mathsf{tl} \circ \mathsf{tl} \circ f_\beta)(x) \\ &= (\mathsf{hd} \circ f_\beta \circ \beta_1)(x) & &= (\mathsf{hd} \circ \mathsf{tl} \circ f_\beta \circ \beta_1)(x) \\ &= (\mathrm{id}_\mathcal{L} \circ \beta_0 \circ \beta_1)(x) & &= (\mathsf{hd} \circ f_\beta \circ \beta_1 \circ \beta_1)(x) \\ &= \beta_0(\beta_1(x)) & &= (\mathrm{id}_\mathcal{L} \circ \beta_0 \circ \beta_1 \circ \beta_1)(x) \\ & & &= \beta_0(\beta_1(\beta_1(x))) \end{aligned}$$

More generally, for every $i \in \mathbb{N}$, we have $l_i = \beta_0(\beta_1^i(x))$. Here, β_1^i stands for repeated application of the function β_1. Now, the function

$$f_\beta : x \mapsto (l_0 l_1 l_2 \ldots), \qquad \text{where} \qquad l_i = \beta_0(\beta_1^i(x))$$

is exactly the function we need to show that the structure z for $\mathcal{L}^\omega$ is final.
We refer to the unique $f_\beta(x) = (l_0 l_1 l_2 \ldots)$ as the stream that x produces.

The coinductive extension maps the behaviour of a B-coalgebra to the final coalgebra. Behavioural equivalence between two coalgebras is obtained by looking at the behaviour in the final coalgebra, by use of the coinductive extensions:

Definition 2.7. Assuming a final B-coalgebra (Z, z) exists, let (X, β) and (Y, γ) be B-coalgebras and let f_β and f_γ be their coinductive extensions to the final coalgebra (Z, z).

Behavioural equivalence between β and γ is the pullback of the cospan $X \xrightarrow{f_\beta} Z \xleftarrow{f_\gamma} Y$ in $\mathbb{C}$. ■

The fact that behavioural equivalence is part of a pullback square means that it is a span $X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$ in $\mathbb{C}$ and the following diagram is a pullback:

$$\begin{tikzcd} R \arrow[r, "\pi_2"] \arrow[d, "\pi_1"] & Y \arrow[d, "f_\gamma"] \\ X \arrow[r, "f_\beta"] & Z \end{tikzcd}$$

Note that π_1 and π_2 need not be morphisms of coalgebras.

Behavioural equivalence between a B-coalgebra (X, β) and itself, is simply referred to as behavioural equivalence of (X, β).

If a final coalgebra does not exist, then (Z,z) in Definition 2.7 can be replaced by a *simple coalgebra.* [1] All results in this section with behavioural equivalence defined for a final coalgebra will then still be valid [25].

In **Set**, behavioural equivalence of two coalgebras (X,β) and (Y,γ) is given by the kernel relation of f_β and f_γ:

$$R := \{(x,y) \in X \times Y \mid f_\beta(x) = f_\gamma(y)\}. \tag{3}$$

For any two functions $X \xleftarrow{\rho_1} S \xrightarrow{\rho_2} Y$ such that $f_\beta \circ \rho_1 = f_\gamma \circ \rho_2$, defining $n : S \longrightarrow R$ by $n : s \mapsto (\rho_1(x), \rho_2(y))$ shows that R is indeed the pullback.

It is often useful to know what behavioural equivalence means in the context of the particular system that the coalgebra models:

Example 2.8. Let $(X,\mathcal{L},\delta)$ and $(Y,\mathcal{L},\epsilon)$ be stream systems. Note that δ and ϵ are L-coalgebras with $L = \mathcal{L}\times(-)$. Recall from Example 2.6 that $(\mathcal{L}^\omega, z)$ with $z = (\mathsf{hd}, \mathsf{tl})$ is the final coalgebra.
Two states $x \in X$ and $y \in Y$ are behaviourally equivalent iff they produce the same stream. Indeed, if $(x,y) \in R$ with R as in (3), then $(f_\delta \circ \pi_1)(x,y) = (f_\epsilon \circ \pi_2)(x,y)$, which means that $f_\delta(x) = f_\epsilon(y)$.

Other examples of coalgebras, and a central topic of this thesis, are *labeled transition systems*.

Definition 2.9. A *labeled transition system* (LTS) is a triple $(X,\mathcal{L},\rightarrow)$, where X is the set of states, $\mathcal{L}$ is a set of labels, and $\rightarrow \subseteq X \times (\mathcal{L} \times X)$ is the transition relation. ■

Like before, we write $x \rightarrow (l,y)$ or $x \xrightarrow{l} y$ whenever $(x,l,y) \in \rightarrow$.

With stream systems, every state has exactly one outgoing transition defined by the transition function δ. Here, the transition relation $\rightarrow$ allows for more than one possible transition, indeed, a *set* of labeled transitions. One state therefore defines not just one stream, but a (possibly infinite) set of traces.

Labeled transition systems are B-coalgebras for the functor $B = \mathcal{P}(\mathcal{L} \times (-))$. An LTS $(X,\mathcal{L},\rightarrow)$ corresponds to a B-coalgebra $\beta : X \longrightarrow \mathcal{P}(\mathcal{L} \times X)$ if we let

$$x \xrightarrow{l} y \iff (l,y) \in \beta(x)$$

This correspondence is then one-on-one by construction of the powerset $\mathcal{P} = 2^{(-)}$ using the subobject classifier, which is the two element set 2 in **Set**. Indeed, the relation $\rightarrow$ is a subset (subobject in **Set**), and hence corresponds to a morphism $\chi_\rightarrow : X \times (\mathcal{L} \times X) \longrightarrow 2$. Through the natural bijection $\hom_{\mathbf{Set}}(X \times (L \times X), 2) \cong \hom_{\mathbf{Set}}(X, 2^{L \times X})$, this in turn corresponds to a morphism $X \longrightarrow \mathcal{P}(L \times X)$, which is our coalgebra structure β.

[1]A B-coalgebra (X,β) is called *simple* if for every B-coalgebra morphism $h : (X,\beta) \longrightarrow (Y,\gamma)$ the following holds: if h is an epimorphism in $\mathbb{C}$, then h is an isomorphism.

Hence, the notion of a labeled transition system and a B-coalgebra for this functor can be used interchangeably.

Example 2.10. We can graphically present an example of an LTS as follows:

$$\begin{tikzcd}
 & & u \arrow[loop right, "a"] & & t'_1 \arrow[r, "b"] & u' \arrow[loop right, "a"] \\
s \arrow[r, "a"] & t \arrow[ur, "b"] \arrow[dr, "c"'] & & s' \arrow[ur, "a"] \arrow[dr, "a"'] & & \\
 & & v \arrow[loop right, "a"] & & t'_2 \arrow[r, "c"'] & v' \arrow[loop right, "a"]
\end{tikzcd}$$

This transition system has a set of states $X = \{s,t,u,v,s',t'_1,t'_2,u',v'\}$ and a set of labels $\mathcal{L} = \{a,b,c\}$.
The coalgebra structure $\beta : X \longrightarrow \mathcal{P}(\mathcal{L}\times X)$ that corresponds to this LTS has, for example, $\beta(t) = \{(b,u),(c,v)\}$ and $\beta(s') = \{(a,t'_1),(a,t'_2)\}$.
Note that s and s' are trace equivalent, as they both admit the same traces $(abaa\ldots)$ and $(acaa\ldots)$. Are they behaviourally equivalent?

To discover what behavioural equivalence means for labeled transition systems, we introduce the notion of a *bisimulation relation*.

Definition 2.11. Let $B : \mathbb{C} \longrightarrow \mathbb{C}$ be a functor. A binary relation $X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$ in $\mathbb{C}$ is a *B-bisimulation* between (X,β) and (Y,γ) if there is a B-coalgebra structure $n : R \longrightarrow BR$ such that the following diagram in $\mathbb{C}$ commutes:

$$\begin{tikzcd}
X \arrow[d, "\beta"] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, dashed, "n"] & Y \arrow[d, "\gamma"] \\
BX & BR \arrow[l, "B\pi_1"] \arrow[r, "B\pi_2"'] & BY
\end{tikzcd}$$

Equivalently, we have a span $(X,\beta) \xleftarrow{\pi_1} (R,n) \xrightarrow{\pi_2} (Y,\gamma)$ in B-**Coalg**. The greatest B-bisimulation, if it exists, is referred to as *B-bisimilarity* and denoted $\sim$. ■

In the case that $\mathbb{C}$ is **Set** and assuming B preserves weak pullbacks, the class of all B-bisimulations between two given B-coalgebras forms a complete lattice, meaning B-bisimilarity exists. [19]

Like with behavioural equivalence (see Example 2.8), it is useful to know what bisimilarity means in the context of the concrete system that a coalgebra represents. (We use $X = Y$ here.)

Definition 2.12. Let $(X,\mathcal{L},\rightarrow)$ be a labeled transition system. A relation $R \subseteq X \times X$ is a *bisimulation for* $(X,\mathcal{L},\rightarrow)$, if xRy implies that, for all $l \in \mathcal{L}$:

(i) for all $x' \in X$, if $x \xrightarrow{l} x'$, then there is $y' \in X$ such that $y \xrightarrow{l} y'$ and $x'Ry'$;

(ii) for all $y' \in X$, if $y \xrightarrow{l} y'$, then there is $x' \in X$ such that $x \xrightarrow{l} x'$ and $x'Ry'$.

Two states $x,y \in X$ are *bisimilar* if there is a bisimulation R such that xRy. ■

The first item says that if xRy, any outgoing labeled transition from x can be mimicked by y, using the same label. The respective targets of the transitions (here x' and y') should again be in the relation. Bisimilarity (the greatest bisimulation) is symmetric by merit of the second item.

It is customary to prove that bisimilarity of labeled transition systems corresponds to the B-bisimilarity.

Lemma 2.13. Bisimilarity for labeled transition systems as in Definition 2.12 corresponds to B-bisimilarity for their corresponding coalgebra structure. ■

Proof. Omitted, but quite straightforward by spelling out the definition of coalgebraic bisimulation in terms of the labeled transition system. See [9] for an example. ☺

Example 2.14. The states s and s' in Example 2.10 are not bisimilar. Indeed, if they were, then there is a bisimulation R such that sRs', and since $s \xrightarrow{a} t$, we must have either tRt'_1 or tRt'_2 (or both).
But tRt'_1 cannot be, since $t \xrightarrow{c} v$, and there is no outgoing transition from t'_1 labeled with c. Similarly, tRt'_2 cannot be, since t'_2 cannot mimic $t \xrightarrow{b} u$.
We conclude that (s,s') cannot be in a bisimulation relation, and hence, s and s' are not bisimilar.

To conclude this section about coalgebras, we give a condition under which behavioural equivalence of a B-coalgebra corresponds to B-bisimilarity. [3]

Lemma 2.15. Assume $\mathbb{C}$ has all pullbacks and let B be an endofunctor on $\mathbb{C}$ that preserves weak pullbacks. Let (X,β) and (Y,γ) be B-coalgebras and assume a final coalgebra (Z,z) exists. Behavioural equivalence between X and Y corresponds to B-bisimilarity. ■

Proof. Let $X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$ be behavioural equivalence. We want to show that any bisimulation relation S is contained in R and furthermore, that R itself is a bisimulation relation.

Let $(X,\beta) \xleftarrow{\rho_1} (S,n) \xrightarrow{\rho_2} (Y,\gamma)$ be a bisimulation relation. Note that ρ_1 and ρ_2 are jointly monic (the pairing $\langle \rho_1, \rho_2 \rangle$ is a monomorphism), and so are π_1 and π_2.

Consider the unique coinductive extensions f_β, f_γ, and f_n from β, γ, and n, respectively, to the final coalgebra z. Since all three are unique, we must have that $f_\beta \circ \rho_1 = f_n$ and $f_\gamma \circ \rho_2 = f_n$, and so we have $f_\beta \circ \rho_1 = f_\gamma \circ \rho_2$.

Since R is a pullback, we get a morphism q from S to R:

$$\begin{tikzcd}
S \arrow[dr, dashed, "q"] \arrow[drr, bend left, "\rho_2"] \arrow[ddr, bend right, "\rho_1"'] & & \\
& R \arrow[r, "\pi_2"] \arrow[d, "\pi_1"] & Y \arrow[d, "f_\gamma"] \\
& X \arrow[r, "f_\beta"] & Z
\end{tikzcd}$$

This shows that $S \leq R$ as subobjects of $X \times Y$ (the associated monomorphisms being the pairings (ρ_1, ρ_2) and (π_1, π_2)). Hence, S is contained in R.

Conversely, to show that the behavioural equivalence R is a bisimulation relation, we use that f_β and f_γ are morphisms of B-coalgebras to obtain

$$\begin{tikzcd}
& R \arrow[dl, "\pi_1"'] \arrow[dr, "\pi_2"] & \\
X \arrow[dr, "!f_\beta"] \arrow[dd, "\beta"] & & Y \arrow[dl, "!f_\gamma"'] \arrow[dd, "\gamma"] \\
& Z \arrow[dd, "z"] & \\
BX \arrow[dr, "Bf_\beta"] & & BY \arrow[dl, "Bf_\gamma"'] \\
& BZ &
\end{tikzcd}$$

as a commuting diagram. Now apply B to the upper (pullback) square:

$$\begin{tikzcd}
BR \arrow[r, "B\pi_2"] \arrow[d, "B\pi_1"] & BY \arrow[d, "Bf_\gamma"] \\
BX \arrow[r, "Bf_\beta"] & BZ
\end{tikzcd}$$

which is a weak pullback square, since B preserves weak pullbacks.

Now use the universal property with $BX \xleftarrow{\beta \circ \pi_1} R \xrightarrow{\gamma \circ \pi_2} BY$ of this weak pullback (note that $Bf_\beta \circ \beta \circ \pi_1 = Bf_\gamma \circ \gamma \circ \pi_2$) to get a (not necessarily unique) morphism n as in the commuting cube

$$\begin{tikzcd}
R \arrow[rr, "\pi_2"] \arrow[dr, dashed, "n"] \arrow[dd, "\pi_1"] & & Y \arrow[dr, "\gamma"] \arrow[dd, "f_\gamma"] & \\
& BR \arrow[rr, "B\pi_2"] \arrow[dd, "B\pi_1"] & & BY \arrow[dd, "Bf_\gamma"] \\
X \arrow[rr, "f_\beta"] \arrow[dr, "\beta"'] & & Z \arrow[dr, "z"] & \\
& BX \arrow[rr, "Bf_\beta"] & & BZ
\end{tikzcd}$$

After some clever diagrammatic origami and omitting z, we can see that n is a witness for R being a bisimulation, so behavioural equivalence is a B-bisimulation itself.

Hence, behavioural equivalence corresponds to B-bisimilarity. ☺

2.2 Algebras for Generating Syntax

We have seen some small examples of labeled stream and transition systems. We also described how they can be viewed as B-coalgebras for a specific endofunctor $B : \mathbb{C} \longrightarrow \mathbb{C}$, where $B = \mathcal{L} \times (-)$ for stream systems, and $B = \mathcal{P}(\mathcal{L} \times (-))$ for transition systems.

The examples we have seen are small, and do not reveal a meaningful interpretation. The states of the systems were just given some seemingly random name. This is not the case in an LTS of a *process calculus*. There, states are interpreted as processes exchanging data and running concurrently. We need a way for these processes to have some sort of shape, which we refer to as the *syntax* of the process calculus. It turns out that syntax can also be described by an endofunctor, which we call S. In this section we will explain *how* the syntax can be described by an endofunctor.

A process calculus often provides a mechanism of writing down states by use of a *syntax grammar*. A syntax grammar consists of rewrite rules, and arises from the notion of an algebraic signature:

Definition 2.16. An *algebraic signature* Σ is a set of *function symbols* or *operators*. To each operator $\mathsf{f} \in \Sigma$ is associated a *term-arity* $\sharp\mathsf{f} \in \mathbb{N}$. ■

Symbols with term-arity zero are referred to as *constants* or *constant symbols*. Operators with arities one, two, or three are referred to as unary, binary, and ternary operators, respectively. For binary operators, we often use infix notations.

If X is some set of term variables, then ΣX is defined as the set of Σ-terms, where only one operator has been applied on elements of X (e.g. $x + y$, with $x, y \in X$ and $+ \in \Sigma$). The set $\Sigma^* X$ denotes the smallest set that contains X and is closed under application of operators of Σ. The star * denotes that we can recursively apply Σ-operations any number of times. Thus, $\Sigma^* X$ contains all terms recursively built from Σ-operations over elements from X.

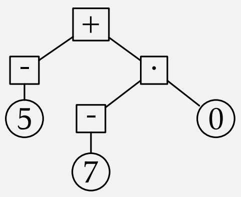
Algebraic signatures correspond to syntax grammars by letting each operator of the signature correspond to a rewrite rule in the grammar. The recursive applications of operators on constants or term variables can be represented by a *syntax tree*. If every branch of that tree ends in a constant then we call it a *closed* Σ-term. Hence, $\Sigma^*\emptyset$ is the set of closed Σ-terms, since it is free of variables.

Example 2.17. Simple arithmetic expressions involving addition, multiplication, and an additive inverse for natural numbers can be generated by an algebraic signature $\Sigma = \{+, \cdot, -\} \cup \mathbb{N}$, with $\sharp + = \sharp \cdot = 2$, $\sharp - = 1$ and $\sharp n = 0$ for all $n \in \mathbb{N}$.
A syntax grammar can be as follows:

$$\mathsf{e} \quad ::= \quad \mathsf{e} + \mathsf{e} \quad | \quad \mathsf{e} \cdot \mathsf{e} \quad | \quad -\mathsf{e} \quad | \quad n \quad | \quad x$$

with $n \in \mathbb{N}$ and $x \in X$ is a term variable. This grammar qualifies any natural number as a closed expression (a constant symbol), but also, e.g., the closed expression $5 + 7 \cdot -0$.
The expression $42 + x$ is *not* closed.

It is often good to specify which operators bind stronger. With arithmetic expressions, for example, it is customary to let negation bind strongest, and multiplication stronger than addition. This means that $-5+-7\cdot 0$ should be interpreted as $(-5)+((-7)\cdot 0)$. The syntax tree of this expression is



Example 2.18. A simplified version of the Calculus of Communicating Systems (CCS) models processes using the following grammar:

$$P \quad ::= \quad P+P \quad | \quad P \,||\, P \quad | \quad \alpha \,.\, P \quad | \quad \mathsf{nil}$$

where α ranges over a set $\mathcal{L}$. We let $\mathcal{L} = \mathbb{A} \cup \overline{\mathbb{A}} \cup \{\tau\}$, with $\mathbb{A}$ some countable infinite set of names, $\overline{\mathbb{A}} = \{\overline{a} \mid a \in \mathbb{A}\}$ that associates to each name a unique dual name, and $\tau \notin \mathbb{A} \cup \overline{\mathbb{A}}$. Furthermore, we let $\overline{\overline{a}} = a$.
We have $\Sigma = \{+, ||, \mathsf{nil}\} \cup \{\alpha \,.\, \_ \mid \alpha \in \mathcal{L}\}$, i.e., one unary operator $\alpha \,.\, \_$ for every $\alpha \in \mathcal{L}$. Note that the grammar contains no rule for a variable, so we only consider closed terms here.
The grammar qualifies e.g. $a \,.\, \overline{b} \,.\, \mathsf{nil}$ as a process (if $a, b \in \mathbb{A}$). The dot operator binds strongest, and $||$ binds stronger than $+$, so the process $a \,.\, \mathsf{nil} \,||\, \overline{a} \,.\, \mathsf{nil} + b \,.\, \mathsf{nil}$ should be understood as $((a \,.\, \mathsf{nil}) \,||\, (\overline{a} \,.\, \mathsf{nil})) + (b \,.\, \mathsf{nil})$.
The interpretation, although not formally specified, should be as follows:

- a process $P+Q$ can behave either as P or as Q;
- a process $P \,||\, Q$ runs P and Q in parallel;
- a process $\alpha \,.\, P$ performs an *action* α and then continues as P;
- a process nil has terminated and cannot perform any actions.

We can study syntax grammars more effectively by viewing them as a categorical construct. Let $S : \mathbb{C} \longrightarrow \mathbb{C}$ be an endofunctor.

Definition 2.19. An *S-algebra* in a category $\mathbb{C}$ is a pair (X, σ), where X is an object of $\mathbb{C}$, called the *carrier*, and $\sigma : SX \longrightarrow X$ is a morphism. ■

We will sometimes write σ in lieu of (X, σ).

To see the correspondence between the syntax generated by an algebraic signature Σ and S-algebras, define the syntax functor

$$SX = \coprod_{\mathsf{f} \in \Sigma} \underbrace{X \times \ldots \times X}_{\sharp\mathsf{f}} \tag{4}$$

An S-algebra structure then is a morphism $\sigma : \coprod_{\mathsf{f}\in\Sigma} X^{\sharp\mathsf{f}} \longrightarrow X$, where each component $X^{\sharp\mathsf{f}} \longrightarrow X$ represents the operation of a single symbol f. We will use Σ and S interchangeably.

For a given endofunctor S, the collection of S-algebras forms a category, denoted S-**Alg**. A morphism between S-algebras σ and τ is a morphism $f : X \longrightarrow Y$ of its underlying carriers X, Y that respects the algebra structure. That is, it makes the following diagram commute:

$$\begin{tikzcd} SX \arrow[r, "Sf"] \arrow[d, "\sigma"] & SY \arrow[d, "\tau"] \\ X \arrow[r, "f"] & Y \end{tikzcd}$$

There is an obvious forgetful functor U_S for every S from S-**Alg** to $\mathbb{C}$.

Initial S-algebras and *final S-algebras* are respectively initial and final objects of the category S-**Alg**. If $\mathbb{C}$ has a final object Z, then the final S-algebra is trivially this object, along with the unique morphism from SZ to it.

If there is an initial algebra, then for any algebra σ, there is a unique morphism g_σ of S-algebras from the initial algebra to σ. The S-algebra morphism g_σ is called the *inductive extension* of σ.

Dual to Lemma 2.5, we have the following lemma to be used later:

Lemma 2.20. Let $S : \mathbb{C} \longrightarrow \mathbb{C}$ be an endofunctor on $\mathbb{C}$. A final object Z of $\mathbb{C}$ yields a unique algebra structure $y : SZ \longrightarrow Z$ which is a final S-algebra. ■

Proof. Dual to the proof of Lemma 2.5. ☺

Example 2.21. Let S be the endofunctor such that $SY = 1 + Y$ with $1 = \{*\}$. The set of natural numbers $\mathbb{N}$ carries an initial algebra structure $\alpha : 1 + \mathbb{N} \longrightarrow \mathbb{N}$ with

$$\alpha : \quad \begin{cases} \iota_1(*) \mapsto 0 \\ \iota_{\mathbb{N}}(n) \mapsto n + 1 \end{cases}$$

Here, ι_1 and $\iota_{\mathbb{N}}$ respectively denote the injections of 1 and $\mathbb{N}$ into $1 + \mathbb{N}$.
To show that α is an initial S-algebra, let (X, σ) be an S-algebra. We need a function $g : \mathbb{N} \longrightarrow X$, such that

$$\begin{tikzcd} 1 + \mathbb{N} \arrow[r, "\mathrm{id}_1 + g"] \arrow[d, "\alpha"] & 1 + X \arrow[d, "\sigma"] \\ \mathbb{N} \arrow[r, "g"'] & X \end{tikzcd}$$

commutes. Put $g : 0 \mapsto \sigma(\iota_1(*))$, since only then $(g \circ \alpha)(\iota_1(*)) = (\sigma \circ \mathrm{id}_1)(\iota_1(*))$. Furthermore, for $n > 0$, put $g : n \mapsto \sigma((g(n-1)))$, since only then

$$(g \circ \alpha)(\iota_{\mathbb{N}}(n)) = g(n+1) = \sigma(g(n)) = (\sigma \circ g)(\iota_{\mathbb{N}}(n))$$

This recursively, totally, and uniquely defines an inductive extension $g_\sigma = g$, and thus, $(\mathbb{N}, \alpha)$ is an initial S-algebra.

The following lemma provides a way to swiftly find a carrier for an initial algebra:

Lemma 2.22. If $S : \mathbf{Set} \longrightarrow \mathbf{Set}$ is the endofunctor corresponding to Σ as described in (4), then the set $\Sigma^*\emptyset$ of closed Σ-terms carries an initial S-algebra structure. ■

Proof. See Appendix B.1. ☺

Example 2.23. An algebra structure for the syntax functor corresponding to the algebraic signature $\Sigma = \{+, \|, \mathsf{nil}\} \cup \{\alpha\ .\ \_ \mid \alpha \in \mathcal{L}\}$ with the syntax grammar

$$P \quad ::= \quad P + P \quad | \quad P \,\|\, P \quad | \quad \alpha\ .\ P \quad | \quad \mathsf{nil}$$

in Example 2.18 is of the form

$$\sigma : X^2 + X^2 + \underbrace{X + \ldots + X}_{\alpha \in \mathcal{L}} + 1 \longrightarrow X$$

The first component $X^2 \longrightarrow X$ is the representation of the symbol $+$, the second for $\|$, and the component $1 \longrightarrow X$ is for the symbol nil. All components $X \longrightarrow X$ represent some operator $\alpha\ .\ \_$ for $\alpha \in \mathcal{L}$.
Since $\mathcal{L}$ is countable, $X + \ldots + X$, as a set, is isomorphic to $\mathcal{L} \times X$. We obtain

$$\sigma : X^2 + X^2 + \mathcal{L} \times X + 1 \longrightarrow X,$$

which may be more intuitive.

As described in Section 1.2, bisimilarity ideally is a congruence. This means that, as a relation, it should respect the S-algebra structure. Formally:

Definition 2.24. Let (X, σ_X) and (Y, σ_Y) be S-algebras. A relation $X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$ in $\mathbb{C}$ is called an *S-congruence* if there is a morphism m such that the following diagram commutes:

$$\begin{tikzcd}
SX \arrow[d, "\sigma_X"] & SR \arrow[l, "S\pi_1"'] \arrow[r, "S\pi_2"] \arrow[d, dashed, "m"] & SY \arrow[d, "\sigma_Y"] \\
X & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & Y
\end{tikzcd}$$

■

If a congruence R is moreover reflexive, and $\mathsf{f} \in \Sigma$ is such that $\sharp\mathsf{f} > 0$, then for every $e_1, \ldots, e_{n-1} \in \Sigma^*\emptyset$, the implication

$$\forall x, x' \in X\ .\ xRx' \implies \mathsf{f}(e_1, \ldots, x, \ldots, e_{\sharp\mathsf{f}-1})\ R\ \mathsf{f}(e_1, \ldots, x', \ldots, e_{\sharp\mathsf{f}-1})$$

holds. Here, the arguments x and x' can be used as the i-th operand for any $i \in \{1, \ldots, \sharp\mathsf{f}\}$, as long as they are on the same place on both sides of the relation.

Example 2.25. Let Σ be as in Example 2.18 and consider the initial algebra structure

$$\sigma : \Sigma\Sigma^*\emptyset \longrightarrow \Sigma^*\emptyset$$

If $R \subseteq \Sigma^*\emptyset \times \Sigma^*\emptyset$ is a congruence, then

$$\begin{tikzcd}
\Sigma\Sigma^*\emptyset \arrow[d, "\sigma"] & \Sigma R \arrow[l, "S\pi_1"'] \arrow[r, "S\pi_2"] \arrow[d, dashed, "m"] & \Sigma\Sigma^*\emptyset \arrow[d, "\sigma"] \\
\Sigma^*\emptyset & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & \Sigma^*\emptyset
\end{tikzcd}$$

commutes. Let $(x_1, x_2), (y_1, y_2) \in R$. Then $(x_1, x_2) + (y_1, y_2)$ is a term in ΣR. Let $(z_1, z_2) \in R$ be the image of this term under m. The commuting diagram then says that $z_1 = x_1 + y_1$ and $z_2 = x_2 + y_2$ and therefore that $(x_1 + y_1)R(x_2 + y_2)$.
We also have $xRx' \implies (\alpha \,.\, x)R(\alpha \,.\, x')$ for any $\alpha \in \mathcal{L}$ and $xRx' \implies (x \parallel y)R(x' \parallel y)$ for all y.

2.3 Stream and Transition System Specifications

We have seen that stream systems and transition systems are coalgebras, and, syntactically, their states can be generated using algebras. The algebras and coalgebras describe a more abstract structure than specific syntax grammars and transition rules of a system.

The interplay between syntax and transition rules becomes visible in *transition system specifications*, which will later be seen to be the combination of algebras and coalgebras into *bialgebras*:

Definition 2.26 (*Transition system specification in GSOS format*)**.** Let Σ be an algebraic signature and $\mathcal{L}$ a set of labels. A *transition system specification* over $(\Sigma, \mathcal{L})$ is a set $\mathcal{R}$ of inference rules, which are expressions of the form

$$\frac{\{\mathtt{x}_{i_j} \xrightarrow{l_j} \mathtt{y}_j\}_{j\in[1,m]}}{\mathsf{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{l} t}$$

in which we have

- a set of *meta-variables* $\mathcal{V} = \{\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{y}_1, \ldots, \mathtt{y}_m\}$, ($i_j \in [1, n]$ for every $j \in [1, m]$)
- a symbol $\mathsf{f} \in \Sigma$ operating on meta-variables $\{\mathtt{x}_1, \ldots \mathtt{x}_n\}$, transitioning to
- a Σ-term $t \in \Sigma^*\mathcal{V}$ built over the set of meta-variables $\mathcal{V}$, and
- labels $l_1, \ldots, l_m, l \in \mathcal{L}$.

■

We could have had a much more general format for inference rules. Consider, for example, the source of the conclusion $\mathsf{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n)$. A more general format would allow this term to be of any shape, just as $t \in \Sigma^*\mathcal{V}$ is of a general shape. Instead, only one operator of Σ can be used once, and the set of meta-variables on which it operates it limited to $\{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$. Another restriction is that the source of every conclusion (the $\mathtt{x}_{i_j}$) must also occur in the source of the conclusion. These restrictions on the shape of the rule are known as the *GSOS rule format* [4]. Later, it will become technically clear why the restrictions are useful.

Note that the set of meta-variables, ranged over by $\mathtt{x}, \mathtt{y}, \ldots$, are *not* the same as variables $x, y, \ldots$ for the states. (Notice here the difference in typesetting.) Instead, they are to be thought of as placeholders for variables or states.

In order to replace the meta-variables by states over a set of state variables X, we require a *substitution*, which is a mapping $\phi : \mathcal{V} \longrightarrow \Sigma^*X$. This substitution can easily be extended to work on Σ-terms over $\mathcal{V}$, by applying it component-wise. We then obtain $\phi : \Sigma^*\mathcal{V} \longrightarrow \Sigma^*X$, so that ϕ can also be applied to t, and so, on all literals of an inference rule. A *closed* substitution is one that maps meta-variables to closed Σ-terms, i.e., $\phi : \mathcal{V} \longrightarrow \Sigma^*\emptyset$.

Finally, we can use inference rules in combination with substitutions to construct a transition system, and deduce valid transitions in our system:

Definition 2.27. Let Σ be an algebraic signature, $\mathcal{L}$ a set of labels, and $\mathcal{R}$ a TSS over $(\Sigma, \mathcal{L})$. Furthermore, let X be a set of state variables. A *proof tree* of a transition $s \xrightarrow{k} s'$ for $\mathcal{R}$ is a finite upwardly branching rooted tree such that

(i) the root $s \xrightarrow{k} s'$ is a transition of terms in Σ^*X and

(ii) if $\{q_{i_j} \xrightarrow{l_{i_j}} r_j\}_{j \in [1,m]}$ is the set of transitions above a node of the tree with transition $p \xrightarrow{l} p'$, then there is a rule

$$\frac{\{\mathtt{x}_{i_j} \xrightarrow{l_j} \mathtt{y}_j\}_{j \in [1,m]}}{\mathsf{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{l} t}$$

and a substitution $\phi : \mathcal{V} \longrightarrow \Sigma^*X$ such that $\phi(\mathtt{x}_{i_j} \xrightarrow{l_j} \mathtt{y}_j) = q_{i_j} \xrightarrow{l_j} r_j$ for all $j \in [1, m]$ and $\phi(\mathsf{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{l} t) = p \xrightarrow{l} p'$.

A transition is *provable* in $\mathcal{R}$ if it is the node of a proof tree. ■

Definition 2.28. Let Σ be an algebraic signature, $\mathcal{L}$ a set of labels, and $\mathcal{R}$ a TSS over $(\Sigma, \mathcal{L})$. A *model* for $\mathcal{R}$ is an LTS $(\Sigma^*X, \mathcal{L}, \rightarrow)$, where X is a set of state variables and the transition relation $\rightarrow$ contains all instances of provable transitions. ■

The intended model of a specification is the smallest model: the model $(\Sigma^*\emptyset, \mathcal{L}, \rightarrow)$ of ground terms, that only uses ground substitutions for instantiating the proof trees.

Example 2.29. Using $\mathcal{L} = \{a, b\}$ and $\Sigma = \{\mathsf{alt}, \mathsf{a}, \mathsf{b}\}$, with $\sharp a = \sharp b = 0$ and $\sharp\mathsf{alt} = 2$, we can define

$$\frac{\mathrm{x} \xrightarrow{l_1} \mathrm{x}' \qquad \mathrm{y} \xrightarrow{l_2} \mathrm{y}'}{\mathsf{alt}(\mathrm{x},\mathrm{y}) \xrightarrow{l_1} \mathsf{alt}(\mathrm{y}',\mathrm{x}')} \qquad \frac{}{\mathsf{a} \xrightarrow{a} \mathsf{a}} \qquad \frac{}{\mathsf{b} \xrightarrow{b} \mathsf{b}}$$

to obtain a very simple stream system where $\mathsf{a} = (aa\ldots)$, $\mathsf{b} = (bb\ldots)$, and, for example, $\mathsf{alt}(\mathsf{a},\mathsf{b}) = (abab\ldots)$ and $\mathsf{alt}(\mathsf{a},\mathsf{alt}(\mathsf{a},\mathsf{b})) = (aaabaaab\ldots)$.
The labels l_1 and l_2 range over $\mathcal{L}$, whereas a and b are fixed.
Note that the rules in this specification adhere to an even stricter format. They are *simple stream SOS rules*: the conclusion target t also has only one operator. Moreover, it is built only from meta-variables that are targets of the premises.

Example 2.30. Let Σ be the algebraic signature and $\mathcal{L} = \mathbb{A} \cup \overline{\mathbb{A}} \cup \{\tau\}$ be the set of labels of CCS defined in 2.18. We can formally give an interpretation to the terms by specifying the following rules:

$$\frac{}{\alpha \,.\, \mathrm{x} \xrightarrow{\alpha} \mathrm{x}}\,\mathsf{Act} \qquad \frac{\mathrm{x} \xrightarrow{\alpha} \mathrm{x}' \qquad \mathrm{y} \xrightarrow{\overline{\alpha}} \mathrm{y}'}{\mathrm{x} \parallel \mathrm{y} \xrightarrow{\tau} \mathrm{x}' \parallel \mathrm{y}'}\,\mathsf{Com}$$

$$\frac{\mathrm{x} \xrightarrow{\alpha} \mathrm{x}'}{\mathrm{x} + \mathrm{y} \xrightarrow{\alpha} \mathrm{x}'}\,\mathsf{SumL} \qquad \frac{\mathrm{y} \xrightarrow{\alpha} \mathrm{y}'}{\mathrm{x} + \mathrm{y} \xrightarrow{\alpha} \mathrm{y}'}\,\mathsf{SumR}$$

$$\frac{\mathrm{x} \xrightarrow{\alpha} \mathrm{x}'}{\mathrm{x} \parallel \mathrm{y} \xrightarrow{\alpha} \mathrm{x}' \parallel \mathrm{y}}\,\mathsf{ParL} \qquad \frac{\mathrm{y} \xrightarrow{\alpha} \mathrm{y}'}{\mathrm{x} \parallel \mathrm{y} \xrightarrow{\alpha} \mathrm{x} \parallel \mathrm{y}'}\,\mathsf{ParR}$$

Figure 1: Set of Generalized Structural Operational Semantics (GSOS) rules $\mathcal{R}_{\mathrm{CCS}}$ for a reduced version of CCS

The specification contains one axiom scheme Act (α ranges over $\mathcal{L}$), by which we get, for example a transition $a \,.\, \mathsf{nil} \xrightarrow{a} \mathsf{nil}$.
We can also prove that the transition $a \,.\, b \,.\, \mathsf{nil} \parallel \overline{a} \,.\, \mathsf{nil} + c \,.\, \mathsf{nil} \xrightarrow{\tau} b \,.\, \mathsf{nil} \parallel \mathsf{nil}$ occurs in the intended model: for the root, consider Rule SumL and the closed substitution that sends x to $a \,.\, b \,.\, \mathsf{nil} \parallel \overline{a} \,.\, \mathsf{nil}$, y to $c \,.\, \mathsf{nil}$, and x' to $b \,.\, \mathsf{nil} \parallel \mathsf{nil}$ to obtain:

$$\frac{a \,.\, b \,.\, \mathsf{nil} \parallel \overline{a} \,.\, \mathsf{nil} \xrightarrow{\tau} b \,.\, \mathsf{nil} \parallel \mathsf{nil}}{a \,.\, b \,.\, \mathsf{nil} \parallel \overline{a} \,.\, \mathsf{nil} + c \,.\, \mathsf{nil} \xrightarrow{\tau} b \,.\, \mathsf{nil} \parallel \mathsf{nil}}\,\mathsf{SumL}$$

So the transition is valid if we can prove $a \,.\, b \,.\, \mathsf{nil} \parallel \overline{a} \,.\, \mathsf{nil} \xrightarrow{\tau} b \,.\, \mathsf{nil} \parallel \mathsf{nil}$.
This is done in the following tree, where some substitutions are silently applied:

$$\frac{\dfrac{}{a \,.\, b \,.\, \mathsf{nil} \xrightarrow{a} b \,.\, \mathsf{nil}}\,\mathsf{Act} \qquad \dfrac{}{\overline{a} \,.\, \mathsf{nil} \xrightarrow{\overline{a}} \mathsf{nil}}\,\mathsf{Act}}{a \,.\, b \,.\, \mathsf{nil} \parallel \overline{a} \,.\, \mathsf{nil} \xrightarrow{\tau} b \,.\, \mathsf{nil} \parallel \mathsf{nil}}\,\mathsf{Com}$$

2.4 Distributing Syntax over Behaviour

We have seen how the behaviour of a transition system is specified by a set of rules, inducing a transition relation that is equivalent to a coalgebra structure. We have also seen how we can use algebraic structures to describe the syntax of states in a transition system.

In this part, we look at distribution of syntax over behaviour through defining a natural transformation $\lambda : SB \Rightarrow BS$.

$$\begin{tikzcd} \mathbb{C} \arrow[loop above, "SB", ""{name=U}] \arrow[loop right, "BS", ""{name=R}] \arrow[loop left, "S"] \arrow[loop below, "B"] \arrow[Rightarrow, from=U, to=R, bend left, "\lambda"] \end{tikzcd} \tag{5}$$

Definition 2.31. A *simple distributive law* of an endofunctor S over an endofunctor B is a natural transformation $\lambda : SB \Rightarrow BS$. ■

Intuitively, SB is a functor that builds a term from different behaviours, and λ derives from this a behaviour for the term, as we can see in the following example:

Example 2.32. Consider Example 2.29, where $\Sigma = \{\mathsf{alt}, \mathsf{a}, \mathsf{b}\}$ and $\mathcal{L} = \{a, b\}$.
For a given X, let $\lambda_X : SBX \longrightarrow BSX$ (where S corresponds to Σ) of be given by the three components

$$\begin{array}{rccccc} \lambda_X^{\mathsf{alt}}: & (\mathcal{L}\times X)^2 & \longrightarrow & \mathcal{L}\times X^2, & (l_1, x', l_2, y') & \mapsto & (l_1, y', x'). \\ \lambda_X^{\mathsf{a}}: & 1 & \longrightarrow & \mathcal{L}\times 1, & \mathsf{a} & \mapsto & (a, \mathsf{a}) \\ \lambda_X^{\mathsf{b}}: & 1 & \longrightarrow & \mathcal{L}\times 1, & \mathsf{b} & \mapsto & (b, \mathsf{b}) \end{array}$$

Taking $\lambda = \lambda_X^{\mathsf{a}} + \lambda_X^{\mathsf{b}} + \lambda_X^{\mathsf{alt}}$, we get a natural transformation $\lambda : SB \Rightarrow BS$:

$$\lambda = \lambda_X^{\mathsf{a}} + \lambda_X^{\mathsf{b}} + \lambda_X^{\mathsf{alt}} \; : \; (\mathcal{L}\times X)^2 + 1 + 1 \longrightarrow \mathcal{L}\times X^2 + \mathcal{L}\times 1 + \mathcal{L}\times 1 \cong \mathcal{L}\times (X^2 + 1 + 1)$$

Notice the similarities with the stream system specification in Example 2.29, where $\mathtt{x}$ (and $\mathtt{y}$) are represented here by streams with head l_1 and tail x' (and head l_2 and tail y'). The image (l_1, y', x') represents an l_1-transition to the term $\mathsf{alt}(y', x')$.

This idea of deriving behaviour of a term is also captured in an inference rule in Definition 2.26: from several behaviours of term variables, we infer behaviour for a term over these variables. Note, however, that our functor S, until now, has been the functor corresponding to Σ, and we cannot yet use several operators of Σ to build a more complex term, as $t \in \Sigma^*\mathcal{V}$ in Definition 2.26 would suggest.

Rather, for an algebraic signature Σ, and with $B = \mathcal{L}\times(-)$, distributive laws $\lambda : SB \Rightarrow BS$ correspond exactly with *simple stream SOS specifications*, which is a specification in a format similar to the GSOS rule format, but much more restrictive. (See [10] for more details on this.) Simple stream SOS rules are of the form

$$\frac{\{\mathtt{x}_i \xrightarrow{l_i} \mathtt{y}_i\}_{i \in \{1..n\}}}{\mathsf{f}(\mathtt{x}_1,\ldots,\mathtt{x}_n) \xrightarrow{l} \mathsf{g}(\mathtt{z}_1,\ldots,\mathtt{z}_{\sharp g})}$$

where all $\mathtt{z}_j \in \{\mathtt{y}_1,\ldots \mathtt{y}_n\}$. There are three big differences with the GSOS rule format:

1. every term variable $\mathtt{x}_i$ under f must occur exactly once as the source of a premise;
2. the conclusion target can only be a term of one operator g, rather than a more complex term $t \in \Sigma^*\mathcal{V}$; and
3. all term variables under g must be targets of the premise.

Note that the stream system specification in Example 2.29 was in this format, but the transition system specification in Example 2.30 was not. It makes it impossible to turn, e.g., the rule ParL into (part of) a natural transformation $SB \Rightarrow BS$ (where S corresponds to Σ), since no outgoing transition for the meta-variable $\mathtt{y}$ has been specified.

It is possible, however, to turn specifications in the GSOS rule format into distributive laws. We will explore this later, in Section 2.6.

2.5 Bialgebras for Congruence of Bisimilarity

Notions of S-algebras for syntax and B-coalgebras for behaviour come together in the notion of a *bialgebra*. For bialgebras, we have that B-bisimilarity is an S-congruence, as we will prove in this section.

Definition 2.33. A *bialgebra* for a distributive law $\lambda : SB \Rightarrow BS$, or λ-bialgebra for short, is a triple (X,σ,β) such that (X,σ) is an S-algebra and (X,β) is a B-coalgebra, and the following diagram commutes:

$$\begin{tikzcd}
SX \arrow[r, "\sigma"] \arrow[d, "S\beta"] & X \arrow[r, "\beta"] & BX \\
SBX \arrow[rr, "\lambda_X"] & & BSX \arrow[u, "B\sigma"']
\end{tikzcd}$$

■

The class of λ-bialgebras forms a category, denoted λ-**bialg**. A bialgebra morphism from (X,σ,β) to (Y,σ,β) is a morphism in $\mathbb{C}$ that is both an algebra morphism and a coalgebra morphism. Note that dependence of λ-**bialg** on S and B is left implicit.

Example 2.34. Let $\lambda_{\Sigma^*\emptyset}$ be as in Example 2.32 with $X = \Sigma^*\emptyset$. We need an algebra structure and a coalgebra structure for a λ-bialgebra with this carrier.
Recall from Lemma 2.22 that $\Sigma^*\emptyset$ carries an initial structure $\alpha : \Sigma\Sigma^*\emptyset \longrightarrow \Sigma^*\emptyset$.

We find $\beta_{\lambda,\alpha}$ such that the following diagram commutes:

$$\begin{tikzcd}
\Sigma\Sigma^*\emptyset \arrow[r, "\alpha"] \arrow[d, "\Sigma\beta_{\lambda,\alpha}=" left, "\mathrm{id}_{\{\mathsf{a}\}}+\mathrm{id}_{\{\mathsf{b}\}}+\beta^2_{\lambda,\alpha}"] & \Sigma^*\emptyset \arrow[r, "\beta_{\lambda,\alpha}"] & \mathcal{L}\times\Sigma^*\emptyset \\
\Sigma(\mathcal{L}\times\Sigma^*\emptyset) \arrow[rr, "\lambda_{\Sigma^*\emptyset}"] & & \mathcal{L}\times\Sigma\Sigma^*\emptyset \arrow[u, "B\alpha=" left, "\mathrm{id}_{\mathcal{L}}\times\alpha"']
\end{tikzcd}$$

This diagram is defined on three coproduct components: {a}, {b}, and {alt}. For the component {a} ({b} is analogous) we have $\mathsf{a} \xmapsto{\Sigma\beta_{\lambda,\alpha}} \mathsf{a} \xmapsto{\lambda^{\mathsf{a}}_{\Sigma^*\emptyset}} (a,\mathsf{a}) \xmapsto{B\alpha^{\mathsf{a}}} (a,\mathsf{a})$, so it easily follows that we need $\beta_{\lambda,\alpha} : \mathsf{a} \mapsto (a,\mathsf{a})$ (and also $\beta_{\lambda,\alpha} : \mathsf{b} \mapsto (b,\mathsf{b})$).
For the operation alt, suppose that, for $x,y \in \Sigma^*\emptyset$, we have $\beta_{\lambda,\alpha} : x \mapsto (l_1,x')$ and $\beta_{\lambda,\alpha} : y \mapsto (l_2,y')$. Then, for alt, we have:

$$\begin{tikzcd}
(x,y) \arrow[r, mapsto, "\alpha"] \arrow[d, mapsto, "\Sigma\beta_{\lambda,\alpha}"'] & (x,y) & (l_1,(y',x')) \\
(l_1,x',l_2,y') \arrow[rr, mapsto, "\lambda^{\mathsf{alt}}_{\Sigma^*\emptyset}"] & & (l_1,y',x') \arrow[u, mapsto, "B\alpha^{\mathsf{alt}}=" left, "(\mathrm{id}_{\mathcal{L}},\alpha^{\mathsf{alt}})"']
\end{tikzcd}$$

If we want a commuting diagram, we must therefore have that $\beta_{\lambda,\alpha} : \mathsf{alt}(x,y) \mapsto (l_1,\mathsf{alt}(y',x'))$.
This inductively defines $\beta_{\lambda,\alpha}$ on all terms of $\Sigma^*\emptyset$.

The coalgebra structure constructed in Example 2.34 seems to be unique. This is indeed the case, and it is because α is initial, as stated in the following lemma:

Lemma 2.35. Let $\lambda : SB \Rightarrow BS$ be a distributive law of S over B.

(i) Let $\alpha : SA \longrightarrow A$ be an initial S-algebra. There is a unique B-coalgebra $\beta_{\lambda,\alpha} : A \longrightarrow BA$ such that $(A,\alpha,\beta_{\lambda,\alpha})$ is a λ-bialgebra. This λ-bialgebra is initial in λ-**bialg**.

(ii) Let $z : Z \longrightarrow BZ$ be a final B-coalgebra. There is a unique S-algebra $\sigma_{\lambda,z} : SZ \longrightarrow Z$ such that $(Z,\sigma_{\lambda,z},z)$ is a λ-bialgebra. This λ-bialgebra is final in λ-**bialg**.

■

The proof requires two functors $B^+ : S$-**Alg** $\longrightarrow S$-**Alg** and $S^+ : B$-**Coalg** $\longrightarrow B$-**Coalg** as the liftings of $B,S : \mathbb{C} \longrightarrow \mathbb{C}$ along the forgetful functors U_S or U^B respectively:

$$\begin{aligned} B^+(X,\sigma) &= (BX, B\sigma\circ\lambda_X) \qquad & S^+(X,\beta) &= (SX,\lambda_X\circ S\beta) \\ B^+f &= Bf & S^+g &= Sg \end{aligned}$$

where f is an S-algebra morphism and g is a B-coalgebra morphism. Note that B^+ and S^+ are implicitly depending on λ.

Is B^+ a functor? We should check that for any S-algebra morphism

$$\begin{tikzcd} SX \arrow[r, "Sf"] \arrow[d, "\sigma_X"] & SY \arrow[d, "\sigma_Y"] \\ X \arrow[r, "f"'] & Y \end{tikzcd}$$

we get that $B^+f = Bf$ is a S-algebra morphism from $B^+(X, \sigma_X) = (BX, B\sigma_X \circ \lambda_X)$ to $B^+(Y, \sigma_Y) = (BY, B\sigma_Y \circ \lambda_Y)$. Apply B to the diagram above and use naturality of λ to combine two commuting squares into one commuting diagram

$$\begin{tikzcd} SBX \arrow[r, "SBf"] \arrow[d, "\lambda_X"] & SBY \arrow[d, "\lambda_Y"] \\ BSX \arrow[r, "BSf"] \arrow[d, "B\sigma_X"] & BSY \arrow[d, "B\sigma_Y"] \\ BX \arrow[r, "Bf"'] & BY \end{tikzcd}$$

showing that Bf is a morphism of S-algebras from $(BX, B\sigma_X \circ \lambda_X)$ to $(BY, B\sigma_Y \circ \lambda_Y)$.

Observe also that B^+ preserves identity and composition, since B does.

In a dual way we can show that S^+ is a functor.

Proof (of Lemma 2.35). We prove item (i). Item (ii) is completely dual.

Let (A, α) be the initial S-algebra as in (i). Note that if (A, α, β) is any λ-bialgebra, then, by Definition 2.33, β is an algebra morphism from (A, α) to $B^+(A, \alpha) = (BA, B\alpha \circ \lambda_A)$:

$$\begin{tikzcd} SA \arrow[r, "\alpha"] \arrow[d, "S\beta"] & A \arrow[r, "\beta"] & BA \\ SBA \arrow[rr, "\lambda_A"] & & BSA \arrow[u, "B\alpha"'] \end{tikzcd}$$

By initiality of α, then, taking $\beta_{\lambda,\alpha}$ as the unique algebra morphism from (A, α) to $B^+(A, \alpha)$ shows existence and uniqueness of the λ-bialgebra $(A, \sigma, \beta_{\lambda,\alpha})$.

We now show that the bialgebra $(A, \alpha, \beta_{\lambda,\alpha})$ is initial in λ-**bialg**. Consider an arbitrary λ-bialgebra (X, σ', β'). By definition, we have that $\beta' : (X, \sigma') \longrightarrow B^+(X, \sigma')$ is an algebra morphism, and therefore a B^+-coalgebra structure in S-**Alg**. Since the B^+-coalgebra $\beta_{\lambda,\alpha} : (A, \alpha) \longrightarrow B^+(A, \alpha)$ is initial by Lemma 2.5, this provides a unique B^+-coalgebra morphism h from $\beta_{\lambda,\alpha}$ to β' (note that h is a morphism in S-**Alg**):

$$\begin{tikzcd} (A, \alpha) \arrow[r, "h"] \arrow[d, "\beta_{\lambda,\alpha}"] & (X, \sigma') \arrow[d, "\beta'"] \\ B^+(A, \alpha) \arrow[r, "B^+h"] & B^+(X, \sigma') \end{tikzcd}$$

Using that $B^+h = Bh$ and unfolding the diagram of S-algebras to a commuting diagram in $\mathbb{C}$, we see that h is an S-algebra morphism that is also a B-coalgebra morphism:

$$\begin{tikzcd}
SA \arrow[r, "\alpha"] \arrow[d, "S\beta_{\lambda,\alpha}"'] \arrow[rrr, bend left, "Sh"] & A \arrow[r, "h"] \arrow[d, "\beta_{\lambda,\alpha}"] & X \arrow[d, "\beta'"'] & SX \arrow[l, "\sigma'"'] \arrow[d, "S\beta'"] \\
SBA \arrow[r, "B\alpha\circ\lambda_A"] \arrow[rrr, bend right, "SBh"'] & BA \arrow[r, "Bh"] & BX & SBX \arrow[l, "B\sigma'\circ\lambda_X"']
\end{tikzcd}$$

So h is a λ-bialgebra morphism and since any bialgebra morphism is also an algebra morphism and (A,α) is initial, h is also unique.

We conclude that $(A,\sigma,\beta_{\lambda,\alpha})$ is initial. ☺

Example 2.36. Let $\lambda_{\mathcal{L}^\omega}$ be as in Example 2.32 with $X = \mathcal{L}^\omega$. In Example 2.34, we built an initial λ-bialgebra. Now we will build a final one.
Recall from Example 2.6 that for the functor $L = \mathcal{L}\times(-)$ a final L-coalgebra structure is given by $z : \mathcal{L}^\omega \longrightarrow \mathcal{L}\times\mathcal{L}^\omega$, defined by $z : l_0l_1l_2\ldots \mapsto (l_0,(l_1l_2\ldots))$, splitting the stream in a *head* l_0 and a *tail* $(l_1l_2\ldots)$.
By Proposition 2.35, we have a final bialgebra with $\sigma_{\lambda,z} : \Sigma\mathcal{L}^\omega \longrightarrow \mathcal{L}^\omega$. Observe that Σ consists of three coproduct components, so $\sigma_{\lambda,z}$ should consist of three components $\sigma^{\mathsf{a}}_{\lambda,z}$, $\sigma^{\mathsf{b}}_{\lambda,z}$, and $\sigma^{\mathsf{alt}}_{\lambda,z}$, such that the diagrams

$$\begin{tikzcd}
\{\mathsf{a}\} \arrow[r, "\sigma^{\mathsf{a}}_{\lambda,z}"] \arrow[d, "Sz"] & \mathcal{L}^\omega \arrow[r, "z"] & \mathcal{L}\times\mathcal{L}^\omega \\
\{\mathsf{a}\} \arrow[rr, "\lambda^{\mathsf{a}}_{\mathcal{L}^\omega}"] & & \mathcal{L}\times\{\mathsf{a}\} \arrow[u, "B\sigma^{\mathsf{a}}_{\lambda,z}="', "(\mathrm{id}_{\mathcal{L}},\sigma^{\mathsf{a}}_{\lambda,z})"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(\mathcal{L}^\omega)^2 \arrow[r, "\sigma^{\mathsf{alt}}_{\lambda,z}"] \arrow[d, "Sz"] & \mathcal{L}^\omega \arrow[r, "z"] & \mathcal{L}\times\mathcal{L}^\omega \\
(\mathcal{L}\times\mathcal{L}^\omega)^2 \arrow[rr, "\lambda^{\mathsf{alt}}_{\mathcal{L}^\omega}"] & & \mathcal{L}\times(\mathcal{L}^\omega)^2 \arrow[u, "B\sigma^{\mathsf{alt}}_{\lambda,z}="', "(\mathrm{id}_{\mathcal{L}},\sigma^{\mathsf{alt}}_{\lambda,z})"]
\end{tikzcd}$$

commute (the diagram for $\sigma^{\mathsf{b}}_{\lambda,z}$ is completely analogous to that of $\sigma^{\mathsf{a}}_{\lambda,z}$).
In the left diagram we see that $\sigma^{\mathsf{a}}_{\lambda,z}$ picks the constant stream $(aa\ldots)$ (and similarly $\sigma^{\mathsf{b}}_{\lambda,z}$ picks $(bb\ldots)$).
In the diagram on the right, considering $((k_0k_1k_2\ldots),(l_0l_1l_2\ldots)) \in (\mathcal{L}^\omega)^2$, compute

$$\begin{aligned}
((k_0k_1k_2\ldots),(l_0l_1l_2\ldots)) &\overset{Sz}{\longmapsto} (l_0,(l_1l_2\ldots),k_0,(k_1k_2\ldots)) \\
&\overset{\lambda^{\mathsf{alt}}_{\mathcal{L}^\omega}}{\longmapsto} (l_0,(k_1k_2\ldots),(l_1l_2\ldots)) \\
&\overset{B\sigma^{alt}_{\lambda,z}}{\longmapsto} (l_0,\sigma^{\mathsf{alt}}_{\lambda,z}((k_1k_2\ldots),(l_1l_2\ldots)))
\end{aligned}$$

Now observe that $\sigma^{\mathsf{alt}}_{\lambda,z}$ maps the pair $((k_0k_1k_2\ldots),(l_0l_1l_2\ldots)) \in (\mathcal{L}^\omega)^2$ to a stream in $\mathcal{L}^\omega$ with head l_0 and tail equal to $\sigma^{\mathsf{alt}}_{\lambda,z}((k_1k_2\ldots),(l_1l_2\ldots))$. Unfolding more 'heads' like this, we see that $\sigma^{\mathsf{alt}}_{\lambda,z}$ picks the *alternating* stream $(l_0k_1l_2k_3\ldots)$.

Assuming an initial S-algebra (A,α) and a final B-coalgebra (Z,z) exist, let $(A,\alpha,\beta_{\lambda,\alpha})$ and $(Z,\sigma_{\lambda,z},z)$ be the initial and final λ-bialgebras for a distributive law $\lambda : SB \Rightarrow BS$. The coinductive extension f_{β_α} of β_α to z and the inductive extension g_{σ_z} from α to σ_z are exactly the same:

$$\begin{tikzcd}
SA \arrow[r, "\alpha"] \arrow[d, "Sk"] & A \arrow[r, "\beta_\alpha"] \arrow[d, dashed, "!k"] & BA \arrow[d, "Bk"] \\
SZ \arrow[r, "\sigma_z"] & Z \arrow[r, "z"] & BZ
\end{tikzcd}$$

This is because there is only one morphism from one to the other. As a consequence, behavioural equivalence on the coalgebra of an initial λ-bialgebra is an S-congruence [10]. This property does not hold for λ-bialgebras in general, unless B preserves weak pullback. See Corollary 2.39 below.

Example 2.37. In Example 2.34 we built an *initial* λ-bialgebra by constructing a B-coalgebra $(\Sigma^*\emptyset, \beta_{\lambda,\sigma})$. Specifically, we made the behaviour for closed Σ-terms explicit.
In Example 2.36, on the other hand, we built a *final* λ-bialgebra by constructing a Σ-algebra $(\mathcal{L}^\omega, \sigma_{\lambda,z})$, where we made explicit the interpretation of streams in $\mathcal{L}^\omega$ under the algebra of operations.
The unique morphism from $\Sigma^*\emptyset$ to $\mathcal{L}^\omega$ has, e.g., $k : \mathsf{alt}(\mathsf{a},\mathsf{b}) \mapsto (abab\ldots)$.
Note that this interpretation of closed Σ-terms arises solely from the distributive law λ specified in Example 2.32 and gives us exactly the intended model of a stream system (a coalgebra) we wanted in Example 2.29.
One might therefore conjecture that specifications and distributive laws are in some sort of correspondence.

The following proposition is a major advantage of studying labeled transition systems in the context of λ-bialgebras:

Proposition 2.38 (*Congruence of Bisimilarity*)**.** Let $\lambda : SB \Rightarrow BS$ be a distributive law. If (X,σ,β) and (Y,τ,γ) are λ-bialgebras, then B-bisimilarity between (X,β) and (Y,γ) is an S-congruence. ■

The following corollary is then easily established with Lemma 2.15:

Corollary 2.39 (*Congruence of Behavioural Equivalence*)**.** Let $\lambda : SB \Rightarrow BS$ be a distributive law. If (X,σ,β) and (Y,τ,γ) are λ-bialgebras and B preserves weak pullbacks, then behavioural equivalence between (X,β) and (Y,γ) is an S-congruence. ■

Proof (of Proposition 2.38). Let R be B-bisimilarity, which is a B-bisimulation itself, and let n be its mediating B-coalgebra structure, i.e., $(X,\beta) \xleftarrow{\pi_1} (R,n) \xrightarrow{\pi_2} (Y,\gamma)$ is a span in B-**Coalg**

Note that π_1 and π_2 are B-coalgebra morphisms, and so, since S^+ is a functor, we get another span of B-coalgebras $S^+(X,\beta) \xleftarrow{S^+\pi_1} S^+(R,n) \xrightarrow{S^+\pi_2} S^+(Y,\gamma)$.

Observe that $\sigma : SX \longrightarrow X$ is a morphism in $\mathbb{C}$ that is also a morphism of B-coalgebras from $S^+(X,\beta) = (SX, \lambda_X \circ S\beta)$ to (X,β), since

$$\begin{tikzcd}
SX \arrow[r, "\sigma"] \arrow[d, "\lambda_X \circ S\beta"] & X \arrow[d, "\beta"] \\
BSX \arrow[r, "B\sigma"] & BX
\end{tikzcd}$$

commutes by merit of (X,σ,β) being a λ-bialgebra. Similarly, $\tau : S^+(Y,\gamma) \longrightarrow (Y,\gamma)$ is a morphism since (Y,τ,γ) is a bialgebra. Thus, we obtain

$$\begin{tikzcd}
(SX, \lambda_X \circ S\beta) \arrow[d, "\sigma"] & (SR, \lambda_R \circ Sn) \arrow[l, "S\pi_1"'] \arrow[r, "S\pi_2"] & (SY, \lambda_Y \circ S\gamma) \arrow[d, "\tau"] \\
(X,\beta) & (R,n) \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & (Y,\gamma)
\end{tikzcd}$$

as a commuting diagram in B-**Coalg**, and this stands witness to $X \xleftarrow{\sigma \circ S\pi_1} SR \xrightarrow{\tau \circ S\pi_2} Y$ being a B-bisimulation for (X,β) and (Y,γ).

Note that both pairings $\langle \pi_1, \pi_2 \rangle$ and $\langle \sigma \circ S\pi_1, \tau \circ S\pi_2 \rangle$ are jointly monic. Furthermore, since R is the largest bisimulation, SR must be contained in R (as subobjects of $X \times Y$ in $\mathbb{C}$). Hence, we have a morphism $m : SR \longrightarrow R$ in $\mathbb{C}$ such that $\langle \pi_1, \pi_2 \rangle \circ m = \langle \sigma \circ S\pi_1, \tau \circ S\pi_2 \rangle$, i.e., we have a commuting diagram

$$\begin{tikzcd}
SX \arrow[d, "\sigma"] & SR \arrow[l, "S\pi_1"'] \arrow[r, "S\pi_2"] \arrow[d, dashed, "m"] & SY \arrow[d, "\tau"] \\
X & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & Y
\end{tikzcd}$$

This shows that R is an S-congruence. ☺

2.6 Abstract Rules

In Section 2.4, we explained how simple stream SOS specifications correspond to distributive laws. Since this kind of specification does not allow us to be very expressive, we explain here how GSOS specifications (as in Definition 2.26) also correspond to distributive laws. This is very fruitful, as it will guarantee that these specifications induce a well-defined transition system that moreover satisfies congruence of bisimilarity.

Recall the definitions of pointed, copointed, and monadic functors. (See also Appendix A.4.) A *copointed coalgebra* for a copointed functor (B,ϵ) with $\epsilon : B \Rightarrow \mathrm{Id}$ is a B-coalgebra (X,β) such that $\epsilon_X \circ \beta = \mathrm{id}_X$. If $\mathbb{C}$ has finite products, then any functor B gives rise to a copointed functor $\mathrm{Id} \times B$ with ϵ being the first projection. This is called the cofree copointed functor for B. For any endofunctor B, there is a bijective correspondence between B-coalgebras and copointed $(\mathrm{Id} \times B)$-coalgebras.

Dually, we have that a *pointed algebra* (X, σ) for a pointed functor (S, η) with $\eta : \mathrm{Id} \Rightarrow S$ is an S-algebra with $\sigma \circ \eta_X = \mathrm{id}_X$. The unit of the free pointed algebras here is the natural family of coproduct injections $\eta : \mathrm{Id} \Rightarrow \mathrm{Id} + S$. Also, if $\mathbb{C}$ has finite coproducts, for any endofunctor S, there is a bijective correspondence between S-algebras and *pointed* $(\mathrm{Id} + S)$-algebras.

Finally, an *Eilenberg-Moore F-algebra* (X, σ) for a monad (F, μ, η) is an F-algebra such that the diagrams

$$\begin{tikzcd}
 & SX \arrow[d, "\sigma"] \\
X \arrow[ur, "\eta_X"] \arrow[r, "\mathrm{id}_X"'] & X
\end{tikzcd}
\qquad
\begin{tikzcd}
SSX \arrow[r, "\mu_X"] \arrow[d, "S\sigma"] & SX \arrow[d, "\sigma"] \\
SX \arrow[r, "\sigma"] & X
\end{tikzcd}$$

commute. Note that the diagram on the left just says that (X, σ) is a pointed (F, η)-algebra.

If a syntax functor S corresponds to an algebraic signature Σ, then the functor S^* forms a monad, called the *free monad*, with unit η and multiplication μ. Here, $\eta_X : X \longrightarrow S^*X$ gives an interpretation of variables in X as terms, and $\mu_X : S^*S^*X \longrightarrow S^*X$ glues terms built from terms.

There is an isomorphism between S-**Alg** and the category S^*-**Alg** of Eilenberg-Moore algebras for the free monad (S^*, μ, η). The initial algebra for the free monad is $S^*\emptyset$.

Definition 2.40. A distributive law of a monad (F, μ, η) over a copointed endofunctor (G, ϵ) is a natural transformation $\lambda : FG \Rightarrow GF$ such that the following diagrams commute:

$$\begin{tikzcd}
FG \arrow[r, Rightarrow, "\lambda"] \arrow[dr, Rightarrow, "F\epsilon"'] & GF \arrow[d, Rightarrow, "\epsilon F"] \\
 & F
\end{tikzcd}
\qquad
\begin{tikzcd}
G \arrow[d, Rightarrow, "\eta G"'] \arrow[dr, Rightarrow, "G\eta"] & \\
FG \arrow[r, Rightarrow, "\lambda"] & GF
\end{tikzcd}
\qquad
\begin{tikzcd}
FFG \arrow[r, Rightarrow, "F\lambda"] \arrow[d, Rightarrow, "\mu G"'] & FGF \arrow[r, Rightarrow, "\lambda F"] & GFF \arrow[d, Rightarrow, "G\mu"] \\
FG \arrow[rr, Rightarrow, "\lambda"] & & GF
\end{tikzcd}$$

■

The first diagram ensures that the coalgebraic structure of λ-bialgebras is copointed, and the other two ensure that the algebras are Eilenberg-Moore algebras.

The result of Proposition 2.38 holds also for distributive laws of monads over copointed functors. However, it is a difficult task to check all the commuting diagrams. Fortunately, we have abstract rules, which correspond to distributive laws and are easier to construct.

Let (S^*, μ, η) be the free monad over an endofunctor S corresponding to an algebraic signature Σ, and (B, ϵ) be a copointed endofunctor over B.

Definition 2.41. An *abstract rule* of S over B is a natural transformation

$$\rho : S(\mathrm{Id} \times B) \Rightarrow BS^*.$$

■

The axiomatic diagrams of Eilenberg-Moore algebras and distributive laws of monads over copointed endofunctors then provide the following result:

Lemma 2.42. Abstract rules of S over B are in a 1-1 correspondence with distributive laws of (S^*, μ, η) over (B, ϵ). ■

Proof. Omitted, but see [11], for example. ☺

As we will show in the example below, GSOS specifications correspond to abstract rules:

Lemma 2.43. Every GSOS specification as in Definition 2.26 gives rise to an abstract rule, and every abstract rule arises from such a GSOS specification. ■

Proof. Also omitted, but see [10], for example. ☺

We can build bialgebras for the distributive law that corresponds to the abstract rule. Those bialgebras then satisfies congruence of bisimilarity. Therefore, the following significant proposition is a corollary of all results in Section 2:

Proposition 2.44. Bisimilarity for transition systems induced by rule specifications in GSOS format are a congruence. ■

Example 2.45. We build an abstract rule for the specification for the simplified CCS given in Example 2.30. Note that, since all the rules are in GSOS format, we already know from Lemmas 2.42 and 2.43 that, through a correspondence with a distributive law, the transition system that the specification induces enjoys congruence of bisimilarity. This example only serves to illustrate in which manner specifications correspond to abstract rules.
The rules in the specification are:

$$\frac{}{\alpha\,.\,\mathsf{x} \xrightarrow{\alpha} \mathsf{x}}\ \mathsf{Act} \qquad \frac{\mathsf{x} \xrightarrow{\alpha} \mathsf{x}' \qquad \mathsf{y} \xrightarrow{\overline{\alpha}} \mathsf{y}'}{\mathsf{x} \parallel \mathsf{y} \xrightarrow{\tau} \mathsf{x}' \parallel \mathsf{y}'}\ \mathsf{Com}$$

$$\frac{\mathsf{x} \xrightarrow{\alpha} \mathsf{x}'}{\mathsf{x} + \mathsf{y} \xrightarrow{\alpha} \mathsf{x}'}\ \mathsf{SumL} \qquad \frac{\mathsf{y} \xrightarrow{\alpha} \mathsf{y}'}{\mathsf{x} + \mathsf{y} \xrightarrow{\alpha} \mathsf{y}'}\ \mathsf{SumR}$$

$$\frac{\mathsf{x} \xrightarrow{\alpha} \mathsf{x}'}{\mathsf{x} \parallel \mathsf{y} \xrightarrow{\alpha} \mathsf{x}' \parallel \mathsf{y}}\ \mathsf{ParL} \qquad \frac{\mathsf{y} \xrightarrow{\alpha} \mathsf{y}'}{\mathsf{x} \parallel \mathsf{y} \xrightarrow{\alpha} \mathsf{x} \parallel \mathsf{y}'}\ \mathsf{ParR}$$

Recall also from Example 2.18 and Example 2.23 that for the associated syntax grammar

$$P \quad ::= \quad P + P \quad | \quad P \parallel P \quad | \quad \alpha\,.\,P \quad | \quad \mathsf{nil},$$

we have a syntax functor S such that

$$SY = Y^2 + Y^2 + \mathcal{L} \times Y + 1.$$

The corresponding coproduct injections we denote by σ_+, $\sigma_\|$, $\sigma_{\mathcal{L}}$, and σ_{nil}. For each injection, we will define part of the abstract rule $\rho : S(\mathrm{Id} \times B) \Rightarrow BS^*$.
For a given X, define

$$\rho^+_X : (X \times \mathcal{P}(\mathcal{L} \times X))^2 \longrightarrow \mathcal{P}(\mathcal{L} \times S^*X), \qquad (x, W_x, y, W_y) \mapsto W_x \cup W_y$$

Here, W_x and W_y are subsets of $\mathcal{L} \times X$ (and also of $\mathcal{L} \times S^*X$) and represent the sets of all possible outgoing transitions for x and y respectively. This part therefore gives all possible outgoing transitions for a term $x + y$, and it combines Rules SumL and SumR.
The part for parallel composition is even more interesting:

$$\begin{array}{rccc} \rho^\|_X : & (X \times \mathcal{P}(\mathcal{L} \times X))^2 & \longrightarrow & \mathcal{P}(\mathcal{L} \times S^*X), \\ & & & \{(\alpha, \sigma_\|(x', y)) \mid (\alpha, x') \in W_x\} \\ & & & \cup \\ & (x, W_x, y, W_y) & \mapsto & \{(\alpha, \sigma_\|(x, y')) \mid (\alpha, y') \in W_y\} \\ & & & \cup \\ & & & \{(\tau, \sigma_\|(x', y')) \mid \exists \alpha \in \mathcal{L} \,.\, (\alpha, x') \in W_x \wedge (\overline{\alpha}, y') \in W_y\} \end{array}$$

Note that all three sets in the domain are an element of $\mathcal{P}(\mathcal{L} \times SX)$, and therefore of $\mathcal{P}(\mathcal{L} \times S^*X)$. The first two sets correspond to Rules ParL and ParR and the third corresponds to Rule Com.
Finally, we have the rules for action prefixing and termination:

$$\begin{array}{lccccc} \rho^{\mathcal{L}}_X : & \mathcal{L} \times X \times \mathcal{P}(\mathcal{L} \times X) & \longrightarrow & \mathcal{P}(\mathcal{L} \times S^*X), & (\alpha, x, W_x) & \mapsto \{(l, x)\} \\ \rho^{\mathsf{nil}} : & 1 & \longrightarrow & \mathcal{P}(\mathcal{L} \times S^*X), & * & \mapsto \emptyset \end{array}$$

Notice how we ignore the set W_x in the rule for prefixing, reflecting the fact that Rule Act is an axiom, and we do not care about the existing outgoing transitions for x. We verify that ρ is indeed natural by checking that

$$\begin{tikzcd} S(X \times BX) \arrow[r, "\rho_X"] \arrow[d, "S(f,Bf)"'] & BS^*X \arrow[d, "BS^*f"] \\ S(Y \times BY) \arrow[r, "\rho_Y"'] & BS^*Y \end{tikzcd}$$

commutes for a morphism $f : X \longrightarrow Y$. We only show this for the component ρ^+, for which we have:

$$\begin{tikzcd} (X \times \mathcal{P}(\mathcal{L} \times X))^2 \arrow[rr, "\rho_X"] \arrow[ddd, "(f,Bf,f,Bf)"'] & & BS^*X \arrow[ddd, "BS^*f"] \\ & (x_1, W_1, x_2, W_2) \arrow[r, mapsto] \arrow[d, mapsto] & W_1 \cup W_2 \\ & (f(x_1), W_1', f(x_2), W_2') \arrow[r, mapsto] & W_1' \cup W_2' \\ (X \times \mathcal{P}(\mathcal{L} \times Y))^2 \arrow[rr, "\rho_Y"'] & & BS^*Y \end{tikzcd}$$

where $W_1' = \{(l, f(x_1')) \mid (l, x_1') \in W_1\}$ and $W_2' = \{(l, f(x_2')) \mid (l, x_2') \in W_2\}$. The diagram commutes, since W_1' and W_2' are exactly the image of W_1 and W_2 (respectively) under $BS^* f$.

3 Nominal Sets

Labeled transition systems of process calculi model processes that create and pass around names (referring to values or variables) with each other. As we explained in Section 1, these models therefore require notions of freshness, α-equivalence, name binding, and capture-avoiding substitutions. In this section we mathematically describe these notions, and this will help us formalize name-passing process calculi with the category theory described in the previous section.

We consider this section as a self-contained and complete stepping stone to the next section. A reader interested in more background theory is advised to consult [18]. There, only G-sets are considered. The theory here is, at times, more general, in that we also describe M-sets. This is needed to define the capture-avoiding substitutions on processes later.

3.1 M-Set and G-Set: Categories of Sets with Actions

We start out with the theory of actions of monoids and groups on sets in general. Later, we will apply all this with the monoid of arbitrary renamings, mapping names to names, and the group of bijective renamings, i.e., the group of permutations of names.

Let X be a set and M a monoid with neutral element e. An *action* of M (or M-action) on X is a function $\cdot : M \times X \longrightarrow X$, such that, using infix notation:

$$\begin{aligned} m \cdot (m' \cdot x) &= (mm') \cdot x \\ e \cdot x &= x \end{aligned}$$

for all $m, m' \in M$ and $x \in X$.

Definition 3.1. If M is a monoid, then an *M-set* is a set X equipped with an action of M on X. ■

For the monoid action of an M-set Y, the same notation is used as the monoid action of a different M-set X. That is, we write both $m \cdot x$ and $m \cdot y$ for $x \in X$ and $y \in Y$, even though the monoid action can be different.

Example 3.2. For a given monoid M, a trivial example of an M-set is the monoid M itself. Its action, called the *natural action*, is the monoid multiplication itself.

The set $\mathrm{End}(X)$ of functions from X to X, described in the following example, is an important monoid:

Example 3.3. Let X be a set. Then the set of functions $\mathrm{End}(X) = \{\delta : X \longrightarrow X\}$ forms a monoid. The monoid operation is function composition and its neutral element is the identity function 1_X. Function application on X turns X into an $\mathrm{End}(X)$-set. Indeed, $1_X \cdot x = 1_X(x) = x$ for all x and

$$\epsilon \cdot (\delta \cdot x) = \epsilon(\delta(x)) = (\epsilon \circ \delta)(x) = (\epsilon \circ \delta) \cdot x$$

For any monoid M, any set X can be made into an M-set using the action $M \times X \longrightarrow X$ given by projection. With this action, X is called a *discrete* M-set.

For a fixed M, the collection of all M-sets forms a category, denoted M-**Set**, where the morphisms are equivariant functions:

Definition 3.4. A function $f : X \longrightarrow Y$ between two M-sets X and Y is called *M-equivariant* (or *equivariant* if M is clear from the context) if it commutes with the monoid action: $f\ (m \cdot x) = m \cdot (f\ x)$ for all $m \in M$ and $x \in X$. ■

Since every group is also a monoid, we can also consider M-sets with M being a group. Any statement we make about M-sets will automatically be true about G-sets. Some results, however, may only hold for G-sets. When we specifically demand that the monoid is a group, we will write G instead of M and talk about G-sets (and G-**Set**) instead of M-sets (and M-**Set**).

The submonoid $\mathrm{Sym}(X)$ of $\mathrm{End}(X)$ containing all bijections forms a group. This group will be an important stepping stone towards construction of the category of nominal sets that we later define.

Example 3.5. For any set X, the monoid $\mathrm{End}(X)$ has a submonoid $\mathrm{Sym}(X) = \{f : X \longrightarrow X \mid f \text{ is a bijection}\}$, which forms a group, called the *symmetric* group on X. As with $\mathrm{End}(X)$, the group operation is function composition and its neutral element is the identity function. The inverse of an element of $\mathrm{Sym}(X)$ is the inverse function.
The action of function application on elements of X turns X into a $\mathrm{Sym}(X)$-set.

3.2 M-Set and G-Set are Cartesian Closed

Fix a monoid M and a group G. In what follows we outline some structures and properties of the categories M-**Set** and G-**Set**. Recall that all properties for M-**Set** also hold for G-**Set**.

The terminal object in M-**Set** is the discrete pointed set $1 = \{*\}$. That is, we have $m \cdot * = *$ for all $m \in M$. The initial object is the empty set, along with the empty M-action.

The product in M-**Set** is constructed just as in **Set**: a product

$$X \times Y = \{(x,y) \mid x \in X, y \in Y\}$$

is an M-set if we let $m \cdot (x,y) = (m \cdot x, m \cdot y)$. We have obvious equivariant projections $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$.

The coproduct in M-**Set**, much like in **Set**, corresponds to disjoint union, i.e.,

$$X + Y = \{0\} \times X \cup \{1\} \times Y.$$

Indeed, $X + Y$ is an M-set if we let $m \cdot (0,x) = (0, m \cdot x)$ and $m \cdot (1,y) = (1, m \cdot y)$. Again, the injections in $X \xrightarrow{\iota_1} X + Y \xleftarrow{\iota_2} Y$ are equivariant functions. (Note that we cannot define an M-action on a union of sets in general.)

The category M-**Set** (and G-**Set**) has products, and also exponentiation:

Lemma 3.6. For any monoid M, the category M-**Set** is cartesian closed. ■

Proof. A constructive proof is given in Appendix B.2. Here, we give a categorical proof.

We can view M as the single object of a category $\mathbb{M}$, where morphisms are the monoid elements, and monoid multiplication mm' is defined as $m' \circ m$. This way, an M-set can be seen as a presheaf on $\mathbb{M}$, i.e., a functor $F : \mathbb{M}^{\mathrm{op}} \longrightarrow \mathbf{Set}$.

Also, for two M-sets FM and GM, an equivariant function *is* a natural transformation $\lambda : F \Rightarrow G$. Indeed, naturality of λ means that the single component λ_M is equivariant. We see that M-**Set** $= [\mathbb{M}^{\mathrm{op}}, \mathbf{Set}]$, the category of functors from $\mathbb{M}^{\mathrm{op}}$ to **Set**.

Since $\mathbb{M}$ is small, $[\mathbb{M}^{\mathrm{op}}, \mathbf{Set}] = M$-**Set** is cartesian closed [12]. ☺

The proof in Appendix B.2 constructs the exponent Y^X as the set of equivariant functions from $M \times X \longrightarrow Y$, using the natural action on M. (See Example 3.2.) The action on Y^X is constructed such that $m \cdot f : (n, x) \mapsto (nm, x)$.

Exponentiation in G-**Set** is constructed exactly as in M-**Set**:

Corollary 3.7. For any group G, the category G-**Set** is cartesian closed. ■

Besides the natural action, another G-action on G is possible. Namely, a conjugation action, defined by:

$$g \odot g' := gg'g^{-1}$$

We use different notation to distinguish from the natural action. Obviously this satisfies the required identity law, and we verify that it is indeed a group action by noting that $g \odot (g' \odot h) = g(g'hg'^{-1})g^{-1} = (gg')h(gg')^{-1} = (gg') \odot h$.

There is a more convenient characterization of an exponent:

Proposition 3.8. Let X and Y be G-sets. Then there is an isomorphism of G-sets

$$Y^X = \hom_{G\text{-}\mathbf{Set}}(G \times X, Y) \cong \hom_{\mathbf{Set}}(X, Y).$$

■

Before proving this statement, the set $\hom_{\mathbf{Set}}(X, Y)$ of (not necessarily G-equivariant!) functions from X to Y needs to be equipped with a G-action: for $f \in \hom_{\mathbf{Set}}(X, Y)$, let $g \cdot f = f_g$, where

$$f_g : X \longrightarrow Y, \quad x \mapsto g \cdot (f(g^{-1} \cdot x))$$

This is indeed a G-action, since $f_e = f$, and

$$\begin{aligned} h \cdot (g \cdot f) = h \cdot f_g : x \mapsto \quad & h \cdot (f_g(h^{-1} \cdot x)) \\ = \quad & h \cdot (g \cdot (f(g^{-1} \cdot h^{-1} \cdot x))) \\ = \quad & hg \cdot (f((hg)^{-1} \cdot x)) \end{aligned}$$

and also

$$(hg) \cdot f = f_{hg} : x \mapsto hg \cdot (f((hg)^{-1} \cdot x))$$

Proof. See Appendix B.3. ☺

3.3 Power Set and Quotients in G-Set

We would also like to have a power object in our new categories, which is what we will explore now.

In **Set**, the power set can be constructed by giving an endofunctor $\mathcal{P} : \mathbf{Set} \longrightarrow \mathbf{Set}$ that maps X to Ω^X, the set of morphisms from X to $\Omega = \{\top, \bot\}$. A subset $S \subseteq X$ then corresponds to its characteristic function $\chi_S : X \longrightarrow \Omega$, where $\chi_S(x) = \top$ if and only if $x \in S$. Let us equip $\mathcal{P}(X)$ with an M-action as follows:

$$m \cdot S = \{m \cdot x \mid x \in S\} \tag{6}$$

Then, indeed, $e \cdot S = S$ and

$$m \cdot (m' \cdot S) = \{m \cdot x' \mid x' \in m' \cdot S\} = \{m \cdot (m' \cdot x) \mid x \in S\} = \{(mm') \cdot x \mid x \in S\} = (mm') \cdot S$$

It is beyond the scope of this thesis to show that the set of subsets of X, equipped with this M-action, is formally the categorical power object of X in M-**Set** (although we know it must exist, since M-**Set** is a Boolean Topos [8, 22]). To our knowledge there is no officially published literature making the power set construction explicit in M-**Set**, but an interested reader may consult [22].

In G-**Set**, things are a little bit clearer: the subobject classifier is the *discrete* two-element G-set Ω [18], meaning that $g \cdot b = b$ for all $g \in G$ and both $b \in \Omega$. A function $\chi \in \Omega^X$ need not be equivariant; see Proposition 3.8. We have a G-action on χ by $(g \cdot \chi) : x \mapsto g \cdot \chi(g^{-1} \cdot x)$ and this corresponds exactly to the G-action on sets (that we defined for M-sets):

$$\chi(x) = g \cdot \chi(e \cdot x) = g \cdot \chi(g^{-1} \cdot g \cdot x) = (g \cdot \chi)(g \cdot x).$$

So if S_χ corresponds to χ then, indeed, $g \cdot x \in g \cdot S$ if and only if $x \in S$.

The power object of a G-set X is now given by $\mathcal{P}X = \Omega^X$. Unlike in **Set**, elements of this power object are not in bijection with subobjects. Subobjects have a special property:

Definition 3.9. An M-set S is an *equivariant subset* of an M-set X if $S \subseteq X$ and for all $m \in M$ and $x \in X$:

$$x \in S \implies m \cdot x \in S.$$

Equivalently, a subset $S \subseteq X$ is equivariant if $m \cdot S \subseteq S$ holds for all $m \in M$. ■

For a G-set X and $S \subseteq X$ an equivariant subset, we can do even better than $g \cdot S \subseteq S$ for all $g \in G$:

$$g \cdot S \subseteq S = e \cdot S = (gg^{-1}) \cdot S = g \cdot (g^{-1} \cdot S) \subseteq g \cdot S.$$

Hence, $S = g \cdot S$ for all $g \in G$ and S equivariant.

Lemma 3.10. Let X be a G-set. Subobjects (in G-**Set**) of X are in bijection with equivariant subsets of X. ■

Proof. Without loss of generality, we can consider a subobject $S \xhookrightarrow{j} X$ as a subset of X, since monomorphisms in G-**Set** are injective equivariant functions [18]. By definition, S is the pullback over $X \xrightarrow{\chi_S} \Omega \xleftarrow{t} 1$, for some unique $\chi_S \in \Omega^X$. But then χ_S is a morphism in G-**Set**, so it must be equivariant. Hence, for all $g \in G$:

$$\chi_S(g \cdot x) = g \cdot \chi_S(x) = \chi_S(x),$$

where the second equality holds because the subobject classifier Ω is discrete. This just means that $x \in S$ if and only if $g \cdot x \in S$, so $g \cdot S = S$ for all g. Hence, S is equivariant. ☺

In any category, relations between objects X and Y are subobjects of the product $X \times Y$. Since the subobjects in G-**Set** are in bijection with equivariant subsets by Lemma 3.10, relations are, by definition, *equivariant* subsets of the product in G-**Set**. To avoid confusion, we will call them *equivariant relations* nonetheless.

Since G-**Set** is cartesian closed and has a subobject classifier $\Omega = 1 + 1$ we can employ the *Equivariance Principle* [18]:

Equivariance Principle. *Functions and relations built from equivariant functions and relations using classical higher-order logic are equivariant.*

An equivalence $\backsimeq$ for a G-set X is an equivariant relation $\backsimeq \subseteq X \times X$ that is reflexive, symmetric and transitive. The quotient map $X \longrightarrow X/_{\backsimeq}$ that maps x to its equivalence class $[x]_{\backsimeq} = \{x' \in X \mid x \backsimeq x'\}$ is equivariant, by equivariance of $\backsimeq$:

$$\forall g \in G \,.\, x \backsimeq x' \implies g \cdot x \backsimeq g \cdot x'.$$

Furthermore, since $\backsimeq$ is equivariant, the quotient set $X/_{\backsimeq}$ inherits an action

$$g \cdot [x]_{\backsimeq} = [g \cdot x]_{\backsimeq}$$

from X, turning $X/_{\backsimeq}$ into a *quotient G-set.*

3.4 Finite Support and Categories of Nominal Sets

Until now the monoid M and the group G have been arbitrary. In defining nominal sets, we are interested in operations on names, for which we use two specific monoids: the first being a monoid of elements acting as arbitrary renamings (sometimes called substitutions). Renamings may be non-injective, sending many names to a single name. The second monoid we are interested in is the group of bijective renamings with elements acting as *permutations,* so every name operation can be inverted. The latter one is the building block for the traditional theory of nominal sets.

For this, first, let $\mathbb{A}$ be a countable infinite set whose elements $a, b, c, \ldots$ are called *atoms* or *names*. Recall that End($\mathbb{A}$) denotes the monoid of endomorphisms of $\mathbb{A}$ as in Example 3.3, and that Sym($\mathbb{A}$) denotes its submonoid of bijections, which forms a group (Example 3.5).

An endomorphism $\sigma \in \mathrm{End}(\mathbb{A})$ is called *finitary* if $\{a \in \mathbb{A} \mid \sigma a \neq a\} \subseteq \mathbb{A}$ is finite. (To avoid heavy use of parentheses in the future, we write σa to mean $\sigma(a)$.) That is, an endomorphism is finitary if it leaves all but finitely many elements unchanged. It is clear that the set of finitary endomorphisms on $\mathbb{A}$ forms a submonoid. We denote this monoid by $\mathrm{Rn}(\mathbb{A})$, and call it the monoid of *renamings*.

If we take the finitary submonoid of $\mathrm{Sym}(\mathbb{A})$ instead of $\mathrm{End}(\mathbb{A})$, we end up with all finitary bijections $\mathrm{Pm}(\mathbb{A})$, which we call the group of *permutations*. An overview of the structures we have just described can be given as follows:

$$\begin{array}{ccc} \mathrm{Rn}(\mathbb{A}) & \prec & \mathrm{End}(\mathbb{A}) \\ \curlyvee & & \curlyvee \\ \mathrm{Pm}(\mathbb{A}) & \prec & \mathrm{Sym}(\mathbb{A}) \end{array}$$

Here, $\prec$ means "*is a submonoid of*". The lower two structures are groups, and the two structures on the left contain only finitary endormorphisms. Those are the ones we are interested in.

We call a finitary renaming *elementary* if it sends one element $a \in \mathbb{A}$ to some $b \in \mathbb{A}$ with $b \neq a$ and leaves all other elements of $\mathbb{A}$ unchanged, and write $[b/a]$ for it.

The counterpart of an elementary renaming for permutations is called a *swapping*, and is written $(b\ a)$. For swappings, we have that $(b\ a) \cdot a = b$ and $(b\ a) \cdot b = a$ and $(b\ a) \cdot c = c$ for all other $c \in \mathbb{A}$ with $a \neq c \neq b$.

Fix the monoid $M = \mathrm{Rn}(\mathbb{A})$ and the group $G = \mathrm{Pm}(\mathbb{A})$. We have two categories:

- $\mathrm{Rn}(\mathbb{A})$-**Set** of $\mathrm{Rn}(\mathbb{A})$-sets, where morphisms are $\mathrm{Rn}(\mathbb{A})$-equivariant functions;
- $\mathrm{Pm}(\mathbb{A})$-**Set** of $\mathrm{Pm}(\mathbb{A})$-sets, where morphisms are $\mathrm{Pm}(\mathbb{A})$-equivariant functions.

Since any action $\mathrm{Rn}(\mathbb{A}) \times X \longrightarrow X$ of an $\mathrm{Rn}(\mathbb{A})$-set can be restricted to $\mathrm{Pm}(\mathbb{A})$, we have a forgetful functor $U_{\mathrm{Pm}} : \mathrm{Rn}(\mathbb{A})\text{-}\mathbf{Set} \longrightarrow \mathrm{Pm}(\mathbb{A})\text{-}\mathbf{Set}$ that maps X to itself with this restricted action. Any $\mathrm{Rn}(\mathbb{A})$-equivariant function is obviously $\mathrm{Pm}(\mathbb{A})$-equivariant, so any morphism can be mapped to itself by U_{Pm}.

Consider some element x of a $\mathrm{Pm}(\mathbb{A})$-set or $\mathrm{Rn}(\mathbb{A})$-set X. We would like to know which elements of $\mathbb{A}$ are relevant for x. If we think of x as a process, we are actually asking what names occur in x.

In the following definition, recall that, for f, g functions with domain X, and $S \subseteq X$, writing $f\,\big|_S = g\,\big|_S$ means that $\forall a \in S \,.\, f(a) = g(a)$.

Definition 3.11. Let X be a $\mathrm{Rn}(\mathbb{A})$-set. A subset $S \subseteq \mathbb{A}$ is a Rn-*support* for an element $x \in X$ if for all $\sigma, \sigma' \in \mathrm{Rn}(\mathbb{A})$:

$$\sigma\,\big|_S = \sigma'\,\big|_S \implies \sigma \cdot x = \sigma' \cdot x.$$

If X is a $\mathrm{Pm}(\mathbb{A})$-set, we range over all permutations $\pi, \pi' \in \mathrm{Pm}(\mathbb{A})$ and call it a Pm-support, or simply *support*. ■

In particular, if we use $\sigma' = \mathrm{id}_{\mathbb{A}}$, then, if S is a Rn-support of x, we have the following implication:

$$(\forall a \in S \,.\, \sigma a = a) \implies \sigma \cdot x = x.$$

In other words, for a support S of x, any renaming that leaves all elements of S unchanged will also leave x unchanged. Intuitively, this means that S contains *at least* all names that "occur" in x. Indeed, the *least support* (which we formally define later) is a generalization of the set of *free variables* occurring in a process.

Recall that our aim is to generalize permutations and renamings on processes of a process calculus. The syntax of these processes consists of finite strings of symbols. Thus, any meaningful renaming or permutation on a process can be done by replacing finitely many names. In other words: the support for a process is finite, and therefore, all elements of the set of processes are finitely supported. This is generalized in the following definition:

Definition 3.12. A *nominal renaming set* is an $\mathrm{Rn}(\mathbb{A})$-set of which every element has some finite Rn-support. A *nominal permutation set* is a $\mathrm{Pm}(\mathbb{A})$-set of which every element has some finite Pm-support. ■

The class of all nominal renaming sets forms a category whose morphisms are just as in $\mathrm{Rn}(\mathbb{A})$-**Set**, i.e., they are $\mathrm{Rn}(\mathbb{A})$-equivariant functions. We denote this category by $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}}$, which is by definition a full subcategory of $\mathrm{Rn}(\mathbb{A})$-**Set**.

Similarly, the class of all nominal permutation sets with $\mathrm{Pm}(\mathbb{A})$-equivariant functions forms a full subcategory $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$ of $\mathrm{Pm}(\mathbb{A})$-**Set**. This is our main category of interest, and we will denote it also by **Nom**. We call its objects *nominal sets* and its morphisms *equivariant functions* (instead of nominal permutation sets and $\mathrm{Pm}(\mathbb{A})$-equivariant functions), whereas objects and morphisms in $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}}$ are still explicitly called nominal renaming sets and $\mathrm{Rn}(\mathbb{A})$-equivariant functions.

Since $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}}$ is a full subcategory of $\mathrm{Rn}(\mathbb{A})$-**Set** and any Rn-support S for a $\mathrm{Rn}(\mathbb{A})$-set is also a Pm-support for a $\mathrm{Pm}(\mathbb{A})$-set, the forgetful functor can be restricted to nominal sets, which we will call $U : \mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}} \longrightarrow \mathbf{Nom}$. The forgetful functor U has a left adjoint [15].

Due to the many categories we just introduced, an overview of their connectedness is given below. The lower two categories are *full* subcategories of the upper two.

$$\begin{tikzcd}
\mathrm{Rn}(\mathbb{A})\text{-}\mathbf{Set} \arrow[r, "U_{\mathrm{Pm}}"] & \mathrm{Pm}(\mathbb{A})\text{-}\mathbf{Set} \\
\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}} \arrow[u, hook] \arrow[r, "U"] & \mathbf{Nom} \arrow[u, hook]
\end{tikzcd}$$

In what follows, we give some useful alternative characterizations of support, where we distinguish between nominal renaming sets and nominal permutations sets:

Lemma 3.13. Let X be a nominal renaming set. A subset $S \subset \mathbb{A}$ is a support for $x \in X$ if and only if, for all renamings $\sigma \in \mathrm{Rn}(\mathbb{A})$:

$$(\forall a \in S \,.\, \sigma a = a) \implies \sigma \cdot x = x.$$

■

Proof. Omitted. See [8]. ☺

For nominal sets (i.e., nominal permutation sets), we have a third characterization:

Lemma 3.14. Let $x \in X$ for some nominal set and $S \subset \mathbb{A}$. The following are equivalent:

(i) S is a support for x;

(ii) for all permutations $\pi \in \mathrm{Pm}(\mathbb{A})$, we have $(\forall a \in S \,.\, \pi a = a) \implies \pi \cdot x = x$;

(iii) $\forall a, b \in \mathbb{A} \setminus S \,.\, (b\ a) \cdot x = x$

■

Proof. Equivalence of (i) and (ii) follows from Lemma 3.13.

For (iii), see Proposition 2.1 in [18]. ☺

Example 3.15. Consider the $\mathrm{Rn}(\mathbb{A})$-set $\mathbb{A}$ itself and let $a \in \mathbb{A}$. In Lemma 3.13, the implication holds trivially for any set S that contains a and for all renamings σ. In particular, $\{a\}$ is a Rn-support for a. Therefore, $\mathbb{A}$ is a nominal renaming set.

Example 3.16. Consider $\mathcal{L} = \mathbb{A} \cup \overline{\mathbb{A}} \cup \{\tau\}$ and the syntax grammar from Examples 2.18 and 2.23, where $\mathbb{A} = \{\overline{a} \mid a \in \mathbb{A}\}$ and $\overline{\overline{a}} = a$ for all $a \in \mathbb{A}$. It is clear (by definition) renamings σ act on elements of $\mathbb{A}$. The actions can trivially be extended to $\mathbb{A} \cup \overline{\mathbb{A}} \cup \{\tau\}$ by $\sigma \cdot \overline{a} := \overline{\sigma \cdot a}$ and $\sigma \cdot \tau = \tau$. Note also that all elements of $\mathcal{L}$ are finitely supported, since every element is just a single name (see Example 3.15). This turns $\mathcal{L}$ into a nominal renaming set.
If we let S be the corresponding syntax functor for the syntax grammar

$$P \quad ::= \quad P + P \quad | \quad P \parallel P \quad | \quad \alpha \,.\, P \quad | \quad \mathsf{nil}$$

then this action can easily be extended to work on closed terms $S^*\emptyset$ (or really over any set of variables), by name-by-name substitution:

$$\begin{aligned} \sigma \cdot (P + Q) &= \sigma \cdot P + \sigma \cdot Q \\ \sigma \cdot (P \parallel Q) &= \sigma \cdot P \parallel \sigma \cdot Q \\ \sigma \cdot (\alpha \,.\, P) &= \sigma \cdot \alpha \,.\, \sigma \cdot Q \\ \sigma \cdot \mathsf{nil} &= \mathsf{nil} \end{aligned}$$

Every renaming action is just a recursive application, with the exception of $\sigma \cdot \alpha$ in the third rule, where actual renaming takes place. This definition gives us a renaming action on the syntax of CCS processes, and since processes contain only a finite amount of names, every element of $S^*\emptyset$ has finite support. This shows that the set $S^*\emptyset$ is a nominal renaming set.
This renaming is exactly the capture-avoiding substitution that we require, although no "captures" are avoided here. This will be done later, when we define name abstraction.

For every nominal (renaming or permutation) set X, there is a function supp_X (or supp if X is clear from the context) that gives the least support (Rn-support or Pm-support), of each element x of X. This set $\mathrm{supp}_X(x)$ is given by the intersection of all finite supports of x. Since all $x \in X$ are finitely supported by definition, $\mathrm{supp}_X(x)$ is always finite.

If X is a nominal permutation or renaming set then $\mathrm{supp}_X(x) \subseteq \mathbb{A}$ is a finite subset for all $x \in X$, and therefore an element of $\mathcal{P}_\omega(\mathbb{A})$. Using the action on subsets defined in (6), then, the function supp_X is equivariant if X is a nominal permutation set. That is, $\pi \cdot \mathrm{supp}(x) = \mathrm{supp}(\pi \cdot x)$ for all $\pi \in \mathrm{Pm}(\mathbb{A})$ and $x \in X$ [18].

For nominal renaming sets, we have a slightly weaker, but still significant result:

Lemma 3.17. Let X be a nominal renaming set. For all $x \in X$ and $\sigma \in \mathrm{Rn}(\mathbb{A})$, we have:

$$\mathrm{supp}(\sigma \cdot x) \subseteq \sigma \cdot \mathrm{supp}(x)$$

■

Proof. Recall that $\sigma \cdot \mathrm{supp}(x) = \{\sigma \cdot x \mid x \in \mathrm{supp}(x)\}$. We show that $\sigma \cdot \mathrm{supp}(x)$ is a support for $\sigma \cdot x$, because then, certainly, it contains the least support $\mathrm{supp}(\sigma \cdot x)$.

Use Definition 3.11 and consider two renamings τ, τ' such that $\tau \big|_{\sigma \cdot \mathrm{supp}(x)} = \tau' \big|_{\sigma \cdot \mathrm{supp}(x)}$. This just means that $\tau(\sigma a) = \tau'(\sigma a)$ for all $a \in \mathrm{supp}(x)$. Hence, $(\tau \circ \sigma) \big|_{\mathrm{supp}(x)} = (\tau' \circ \sigma) \big|_{\mathrm{supp}(x)}$, and by definition of support, then, $(\tau \circ \sigma) \cdot x = (\tau' \circ \sigma) \cdot x$. Thus, $\tau \cdot (\sigma \cdot x) = \tau' \cdot (\sigma \cdot x)$, which was to be shown. We conclude that $\sigma \cdot \mathrm{supp}(x)$ is a support for $\sigma \cdot x$. ☺

The action on some element $x \in X$ can be restricted to the minimal support of x, as stated in the following lemma:

Lemma 3.18. Let X be a nominal renaming set. The following implication holds for all renamings $\sigma, \sigma' \in \mathrm{Rn}(\mathbb{A})$:

$$\sigma \cdot x = \sigma' \cdot x \implies \sigma \big|_{\mathrm{supp}(x)} \cdot x = \sigma' \big|_{\mathrm{supp}(x)} \cdot x.$$

■

Proof. Immediate from Definition 3.11 and the fact that $\mathrm{supp}(x)$ is a support for x. ☺

Finally, we can characterize the support of $\mathrm{Pm}(\mathbb{A})$, which is a $\mathrm{Pm}(\mathbb{A})$-set, as follows:

Lemma 3.19. The $\mathrm{Pm}(\mathbb{A})$-set $\mathrm{Pm}(\mathbb{A})$ with the conjugation action $\pi \odot \pi' = \pi \circ \pi' \circ \pi^{-1}$, is a nominal set, and

$$\mathrm{supp}(\pi) = \{a \in \mathbb{A} \mid \pi a \neq a\}$$

■

Proof. Omitted. See [18]. ☺

3.5 Constructions in the Category of Nominal Sets

We prove that **Nom** is cartesian closed in this section, and we give some other constructions.

Lemma 3.20 (2.12 in [18])**.** Let X, Y be $\mathrm{Pm}(\mathbb{A})$-sets and $X \xrightarrow{f} Y$ an equivariant function. If $A \subseteq \mathbb{A}$ supports $x \in X$, then it supports $f(x)$. Furthermore, if f is injective, then $A \subseteq \mathbb{A}$ supports x if and only if it supports $f(x)$. ■

Corollary 3.21. If X, Y are nominal sets and $X \xhookrightarrow{f} Y$ is equivariant, then

$$\mathrm{supp}_Y(f(x)) \subseteq \mathrm{supp}_X(x),$$

where equality holds if f is injective. ■

Proof (of Lemma 3.20). Use Definition 3.11 for A being a support for x: for all $\pi \in \mathrm{Pm}(\mathbb{A})$, we have

$$(\forall a \in A \,.\, \pi a = a) \implies \pi \cdot x = x \implies f(x) = f(\pi \cdot x) = \pi \cdot f(x),$$

where the last equality holds by equivariance of f, so we see that A supports $f(x)$ too.

If f is injective and A supports $f(x)$, then

$$(\forall a \in A \,.\, \pi a = a) \implies \pi \cdot f(x) = f(x) \implies f(\pi \cdot x) = f(x) \implies \pi \cdot x = x,$$

where injectivity is used in the last implication. This shows that A also supports x. ☺

The following lemma will be important to show that **Nom** is Cartesian closed.

Lemma 3.22. **Nom** is a coreflective subcategory of $\mathrm{Pm}(\mathbb{A})$-**Set**. ■

Proof. We show that the inclusion functor $I : \mathbf{Nom} \hookrightarrow \mathrm{Pm}(\mathbb{A})\text{-}\mathbf{Set}$ has $(-)_{\mathrm{fs}}$ as a right adjoint, where $X_{\mathrm{fs}} := \{x \in X \mid x \text{ has finite support}\}$. Note that $X_{\mathrm{fs}} \subseteq X$ is equivariant: any permutation π is finitary, so $x \in X$ has finite support if and only if $\pi \cdot x$ has finite support. The action on X_{fs} is inherited from X.

To show that $I \dashv (-)_{\mathrm{fs}}$, we show that, for Y a nominal set and X a $\mathrm{Pm}(\mathbb{A})$-set, there is a (natural) isomorphism of sets:

$$\mathrm{hom}_{\mathrm{Pm}(\mathbb{A})\text{-}\mathbf{Set}}(Y, X) \cong \mathrm{hom}_{\mathbf{Nom}}(Y, X_{\mathrm{fs}}),$$

The set Y is nominal, so the finite support $\mathrm{supp}(y)$ for y is also a support of $f(y)$, by Lemma 3.20. Therefore, for the isomorphism we can just take the identity, and naturality then follows easily. ☺

Coproducts and products in **Nom** are just like in $\mathrm{Pm}(\mathbb{A})$-**Set**. Finite products of nominal sets as in $\mathrm{Pm}(\mathbb{A})$-**Set** are again nominal sets: for nominal sets $X_1, \ldots, X_n$, the support of an element $(x_1, \ldots, x_n) \in X_1 \times \ldots \times X_n$ is the union of the supports for $x_1, \ldots, x_n$.

However, elements of infinite products need not be finitely supported. For infinite products of a collection $\{X_i\}_{i \in I}$, the product is given by $(\prod_{i \in I} X_i)_{\mathrm{fs}}$.

Proposition 3.23. **Nom** is Cartesian closed. ■

Proof. We have described the finite product (just as in $\mathrm{Pm}(\mathbb{A})$-**Set**), so it remains to show that **Nom** has exponents. We claim that the exponent in **Nom** is given by restricting the exponent in $\mathrm{Pm}(\mathbb{A})$-**Set** to finitely supported subsets, i.e., there is a natural isomorphism $\mathrm{hom}_{\mathbf{Nom}}(Z \times X, Y) \cong \mathrm{hom}_{\mathbf{Nom}}(Z, (Y^X)_{\mathrm{fs}})$.

Indeed, we have a sequence of natural isomorphisms

$$\hom_{\mathbf{Nom}}(Z \times X, Y) \cong \hom_{\mathrm{Pm}(\mathbb{A})\text{-}\mathbf{Set}}(Z \times X, Y) \cong \hom_{\mathrm{Pm}(\mathbb{A})\text{-}\mathbf{Set}}(Z, Y^X) \cong \hom_{\mathbf{Nom}}(Z, (Y^X)_{\mathrm{fs}}),$$

The first isomorphism is because **Nom** is a full subcategory and finite products are the same in both categories. The second isomorphism is by definition of exponentiation in Pm($\mathbb{A}$)-**Set**. Finally, the last isomorphism holds by Lemma 3.22. ☺

Relations between two objects X and Y in **Nom** are again equivariant subsets of the product $X \times Y$, just as in Pm($\mathbb{A}$)-**Set**. This is because their products coincide, and **Nom** is a full subcategory, so subobjects in **Nom** correspond to subobjects in Pm($\mathbb{A}$)-**Set**.

A power object $\mathcal{P}X$ in **Nom** is given by restricting the power object $\mathcal{P}X$ in Pm($\mathbb{A}$)-**Set** (which contains all subsets) to all finitely supported subsets of X. The power object $\mathcal{P}X$ of a nominal set X is thus given by $(\Omega^X)_{\mathrm{fs}}$. To be explicit, we will write $\mathcal{P}_{\mathrm{fs}}(X)$ for the power objects in **Nom**, and call it the *nominal power set*.

The following lemma for finitely supported subsets will be useful later:

Lemma 3.24. Let X be a Pm($\mathbb{A}$)-**Set**. A subset $A \subseteq \mathbb{A}$ supports an element $S \in \mathcal{P}_{\mathrm{fs}}X$ if and only if for every permutation $\pi \in \mathrm{Pm}(\mathbb{A})$:

$$(\forall a \in A \,.\, \pi a = a) \implies \forall x \in S \,.\, \pi \cdot x \in S$$

■

Proof. Omitted. [18] ☺

Quotient sets in **Nom** are as in Pm($\mathbb{A}$)-**Set**.

Lemma 3.25. If X is a nominal set and $\backsimeq$ is an equivariant equivalence relation, then

$$\mathrm{supp}([x]_{\backsimeq}) = \bigcap_{y \in [x]_{\backsimeq}} \mathrm{supp}(y)$$

for all $x \in X$. ■

Proof. Omitted. [18] ☺

We conclude this section with the *Finite Support Principle* [18], which holds by merit of **Nom** being Cartesian closed and having a subobject classifier:

Finite Support Principle. *Functions and relations built from finitely supported functions and relations using classical higher-order logic are finitely supported.*

3.6 Freshness: Avoid Colliding Names

In this section, we formalize the notion of *freshness*, which, intuitively, expresses that a name does not already occur in an element, e.g., a process.

By writing a *freshness assertion* $x \# y$, with $x \in X$ and $y \in Y$, we mean to say that $\mathrm{supp}_X(x) \cap \mathrm{supp}_Y(y) = \emptyset$.

Example 3.26. Many transition specification rules for name-passing process calculi contain freshness conditions. Consider, e.g., the Rule ParL in Example 2.30:

$$\frac{\mathrm{x} \xrightarrow{\alpha} \mathrm{x}'}{\mathrm{x} \parallel \mathrm{y} \xrightarrow{\alpha} \mathrm{x}' \parallel \mathrm{y}}\ \textsf{ParL}$$

If α represents an input action, written $P \xrightarrow{a(b)} P'$ in a π-calculus, then the name b represents a *new*, local, binding name within the scope of P'.
This means that, if we apply Rule ParL and thus expand the scope of the locally instantiated name b, like we do with Q in $P \parallel Q \xrightarrow{a(b)} P' \parallel Q$, then this name b should not clash with any name already occurring in Q.
We therefore add the freshness condition $b \# \mathrm{y}$ to the rule:

$$\frac{\mathrm{x} \xrightarrow{a(b)} \mathrm{x}' \qquad b \not\approx \mathrm{y}}{\mathrm{x} \parallel \mathrm{y} \xrightarrow{a(b)} \mathrm{x}' \parallel \mathrm{y}}$$

By writing $b \not\approx \mathrm{y}$ in this rule we mean that $b \# \phi(\mathrm{y})$ whenever we apply some ground substitution $\phi : \{\mathrm{x}, \mathrm{x}', \mathrm{y}\} \longrightarrow S^*\emptyset$. The freshness assertion has $b \in \mathbb{A}$ and $\phi(\mathrm{y}) \in S^*\emptyset$, and then $b \# \phi(\mathrm{y})$ says that $\{b\} \cap \mathrm{supp}(\phi(\mathrm{y})) = \emptyset$, which is true if and only if $b \notin \mathrm{supp}(\phi(\mathrm{y}))$.

The way the freshness assertion in Example 3.26 is used is most frequent: for $x \# y$ we often use some $x \in \mathbb{A}$ and y in some nominal set Y, which is true if and only if $x \notin \mathrm{supp}_Y y$. Since Y is a nominal set, whose elements all have finite support, there is *always* an infinite amount of names to choose that are fresh in y.

Lemma 3.27. For $X \xrightarrow{f} Y$ in **Nom** and $a \in \mathbb{A}$ we have:

$$a \# x \implies a \# f(x)$$

■

Proof. Quite immediate from Lemma 3.20. ☺

3.7 Abstraction: Formalize Local Binding Names

In what follows we show how the local instantiation of new names, also called *binding names*, are generalized in **Nom**. For this, we first define the notion of α-equivalence. We use definitions and results from [18] and skip the formal construction (which uses a third quantifier И besides $\exists$ and $\forall$).

Definition 3.28. Let X, Y be nominal sets. We define α-equivalence on $X \times Y$, written $\approx_\alpha$, to be a binary relation such that $(x, y) \approx_\alpha (x', y')$ if and only if

$$\exists \pi \in \mathrm{Pm}(\mathbb{A})\ .\ \pi \cdot (x, y) = (x', y') \wedge \pi \# (\mathrm{supp}_Y(y) \setminus \mathrm{supp}_X(x)) \wedge \mathrm{supp}(\pi) \subseteq \mathrm{supp}(x, x').$$

■

This relation is an equivalence:

- reflexivity follows from taking the identity permutation,
- symmetry follows from taking the inverse permutation, and
- transitivity follows from taking the composition of the two permutations.

We are mostly interested in binding names from $\mathbb{A}$, so for $X = \mathbb{A}^k$ for some $k > 0$. For this, we have $(a_1,\dots,a_k,x) \approx_\alpha (a'_1,\dots,a'_k,x')$ if and only if there is a π such that $\pi \cdot (a_1,\dots,a_k,x) = (a'_1,\dots,a'_k,x')$ and $\pi a \neq a$ implies that $a \notin \mathrm{supp}(y)$ and $a \in \{a_1,\dots,a_k,a'_1,\dots,a'_k\}$. For $k = 1$ in particular, we have the following result, which is a standard characterization of α-equivalence:

Lemma 3.29. For α-equivalence on $\mathbb{A} \times X$, we have

$$(a,x) \approx_\alpha (a',x') \iff (a,x) = (a',x') \vee (a' \,\#\, (a,x) \wedge x' = (a\ a') \cdot x).$$

■

Proof. Note that if $\mathrm{supp}(\pi) \subseteq \{a,a'\}$, then either $\pi = (a\ a')$ for some $a \neq a'$ (meaning $a' \,\#\, a$) or π is the identity. Also, $(a\ a') \,\#\, (\mathrm{supp}(x') \setminus \{a\})$ holds true if and only if $a' \notin \mathrm{supp}(x')$, which is the same as $a' \,\#\, x$. ☺

Definition 3.30. For X a nominal set, the *abstraction* $\left[\mathbb{A}^k\right]X$ of $\mathbb{A}^k$ in X is the quotient of $\mathbb{A}^k \times X$ by $\approx_\alpha$. Elements of $\left[\mathbb{A}^k\right]X$, i.e., equivalence classes $[(a_1,\dots,a_k,x)]_{\approx_\alpha}$ are written $\langle a_1,\dots,a_k\rangle x$. ■

NB: the notation using angled brackets for name abstraction is an unfortunate clash with the notation we used for output of names in the π-calculus that we used in Section 0.

Since **Nom** has quotients, an abstraction $\left[\mathbb{A}^k\right]X$ has a $\mathrm{Pm}(\mathbb{A})$-action which is well-defined as

$$\pi \cdot \langle a_1,\dots,a_k\rangle x = \langle(\pi a_1,\dots,\pi a_k)\rangle(\pi \cdot x)$$

Abstraction $\left[\mathbb{A}^k\right](-)$ is an endofunctor on **Nom** [18].

One can think of elements of an abstraction as terms with a locally instantiated name. Examples of this are the λ-calculus term $\lambda x.M$ (formally $\langle x\rangle M$) or the input prefixed π-calculus term $a(b)\,.\,P$ (formally $(a,\langle b\rangle P)$).

In process calculi, the capture-avoiding substitutions are mappings that need not be injective, so we would also like to have the abstraction endofunctor for nominal renaming sets.

This lifting is known in the literature [8], but, to our knowledge, this is the first time it has been generalized to abstraction of several names.

Lemma 3.31. The endofunctor $\left[\mathbb{A}^k\right](-)$ for $k > 0$ lifts along the forgetful functor $U : \mathbf{Nom}^{\mathrm{Rn}}_{\mathbb{A}} \longrightarrow \mathbf{Nom}$ as follows: for X a nominal renaming set, we define an $\mathrm{Rn}(\mathbb{A})$-action on $\left[\mathbb{A}^k\right]X$ by

$$\sigma \cdot \langle a_1,\dots,a_k\rangle x = \langle a_1,\dots,a_k\rangle(\sigma \cdot x) \quad \text{where} \quad a_1,\dots,a_k \notin W_\sigma.$$

where $W_\sigma = \{a \mid \sigma a \neq a\} \cup \{\sigma a \mid \sigma a \neq a\}$ ■

Notice that we can finally see avoidance of capture formally at work here: the bound names a_i are 'silently renamed' to names that are not in any way involved with σ.

Proof. Let us first check that this $\mathrm{Rn}(\mathbb{A})$-action is well-defined. That is, for X a nominal renaming set, and $\approx_\alpha$ the α-equivalence relation on $\mathbb{A}^k \times UX$, we need to make sure that the following implication holds:

$$(a_1,\ldots,a_k,x) \approx_\alpha (a_1',\ldots,a_k',x') \implies (a_1,\ldots,a_k,\sigma\cdot x) \approx_\alpha (a_1',\ldots,a_k',\sigma\cdot x') \tag{7}$$

whenever $a_1,\ldots,a_k,a_1',\ldots,a_k' \notin W_\sigma$.

So let $(a_1,\ldots,a_k,x) \approx_\alpha (a_1',\ldots,a_k',x')$. We let $A = \{a_1,\ldots,a_k,a_1',\ldots,a_k'\}$. Using Definition 3.28, we get a permutation π for which $\pi a_i = a_i'$ for all $i \in \{1,\ldots,k\}$ and $\pi\cdot x = x'$. Moreover, $\pi \mathrel{\#} (\mathrm{supp}(x) \setminus \{a_1,\ldots,a_k\})$ and $\mathrm{supp}(\pi) \subseteq A$. To prove (7), it suffices to show now that $\pi\cdot(\sigma\cdot x) = \sigma\cdot x'$ and $\pi \mathrel{\#} (\mathrm{supp}(\sigma\cdot x) \setminus \{a_1,\ldots,a_k\})$.

By choice of $a_1,\ldots,a_k,a_1',\ldots,a_k'$ and definition of W_σ, we have:

$$\forall a \in A\,.\, \sigma x = a \iff x = a \tag{8}$$

Recall form Lemma 3.19 that $\mathrm{supp}(\pi) = \{a \in \mathbb{A} \mid \pi a \neq a\}$. Note also that $\mathrm{supp}(\pi) = \mathrm{supp}(\pi^{-1})$ since $\pi a = a \iff a = \pi^{-1} a$.

First, we claim that $\pi\cdot A = A$: indeed, suppose for contradiction that $\pi a \notin A$ for some $a \in A$. Since $A \supseteq \mathrm{supp}(\pi) = \mathrm{supp}(\pi^{-1})$, it follows that $\pi a \notin \mathrm{supp}(\pi^{-1})$. By definition of $\mathrm{supp}(\pi^{-1})$, then, $\pi^{-1}(\pi a) = \pi a$, so $\pi a = a \in A$, but we assumed that $\pi a \notin A$.

Our claim is now that $\sigma \circ \pi = \pi \circ \sigma$.

- If $a \in A$, then $\pi(\sigma a) = \pi a = \sigma(\pi a)$. Both equalities hold by (8), using first that $a \in A$, and then that $\pi a \in \pi\cdot A = A$.
- If $a \notin A$, then $\sigma(\pi a) = \sigma a$. Suppose $\pi(\sigma a) \neq \sigma a$, then $\sigma a = a'$ for some $a' \in A$. By (8), then, $a = a'$, so $a \in A$, which is a contradiction. Therefore, our assumption that $\pi(\sigma a) \neq \sigma a$ was false.

The claim follows from these two cases. From this and $x' = \pi\cdot x$, it follows that

$$\sigma\cdot x' = \sigma\cdot(\pi\cdot x) = (\sigma\circ\pi)\cdot x = (\pi\circ\sigma)\cdot x = \pi\cdot(\sigma\cdot x)$$

To finish the proof of (7), note that if $a \in \mathrm{supp}(\pi)$, then $a \notin (\mathrm{supp}(x) \setminus \{a_1,\ldots,a_k\})$, and $a \in A$, so $\sigma a = a$ by (8). There are two cases to distinguish for $a \notin (\mathrm{supp}(x) \setminus \{a_1,\ldots,a_k\})$:

- If $a \in \mathrm{supp}(x)$, then $a \in \{a_1,\ldots,a_k\}$, and so $a \notin \mathrm{supp}(\sigma\cdot x) \setminus \{a_1,\ldots,a_k\}$.
- If $a \notin \mathrm{supp}(x)$, note that $\sigma\cdot\mathrm{supp}(x) \supseteq \mathrm{supp}(\sigma\cdot x)$ b

Finally, if σ is a permutation, then the choices of $a_1,\ldots,a_k$ furthermore ensure that $\langle a_1,\ldots,a_k\rangle\sigma\cdot x = \langle\sigma a_1,\ldots,\sigma a_k\rangle\sigma\cdot x = \sigma\cdot\langle a_1,\ldots,a_k\rangle x$, showing that the $\mathrm{Pm}(\mathbb{A})$-actions for $\left[\mathbb{A}^k\right]UX$ and $U\left[\mathbb{A}^k\right]X$ coincide.

Hence, $\left[\mathbb{A}^k\right](-)$ lifts along U. ☺

4 Nominal Transition Systems

In Section 2, we outlined the theory underlying transition systems and in Section 3, we described a categorical structure that supports name binding and substitution of names. In this section we explore how these two theories combine into nominal transition systems, where the set of states and the set of labels are nominal sets and the transition relation is equivariant.

Like with transition systems of sets, we aim to find a correspondence between coalgebras and transition systems. As we will see, the challenge herein is to find a structure for the labels of the transition system in such a way that it corresponds to the use of a label functor in the coalgebra. This is because, ideally, the abstraction endofunctor is used in the label functor to model, e.g., input, as input uses binding names.

4.1 Coalgebras in Nom: Nominal Transition Systems

Transition systems of nominal sets can be straightforwardly defined as follows:

Definition 4.1. A *nominal transition system* (NTS) is a triple $(X, \mathfrak{L}, \rightarrow)$ where the carrier X is a nominal set, $\mathfrak{L}$ is a nominal set of labels, and the transition relation $\rightarrow \subseteq X \times (\mathfrak{L} \times X)$ is equivariant.

The quadruple is called a *nominal renaming transition system* if X is a nominal renaming set. ■

We will assume the following structure on the nominal set of labels:

Assumption 4.2. We will assume that $\mathfrak{L}$ is of the form

$$\mathfrak{L} = \coprod_{i=1}^{t} \mathbb{A}^{k_i},$$

for some $t \in \mathbb{N}_{>0}$ and $k_1, \ldots, k_t \in \mathbb{N}_{\geq 0}$. ■

An important remark here is that the transition relation of a nominal *renaming* transition system, by definition, need not be Rn-equivariant. That is, for any permutation $\pi \in \mathrm{Pm}(\mathbb{A})$, if $x \xrightarrow{l} y$ is a transition in some nominal transition system (with or without renaming), then $(\pi \cdot x) \xrightarrow{(\pi \cdot l)} (\pi \cdot y)$ always, whereas this does not hold true in general for renamings $\sigma \in \mathrm{Rn}(\mathbb{A})$.

Nominal renaming transition systems are therefore just nominal transition system, but with a set of states carrying a renaming action. As per Lemma 3.31, this renaming action is the usual capture-avoiding substitution used in many process calculi.

Like in [1, 6, 24], we refer to elements of $\mathfrak{L} \times X$ as *residuals* . A residual thus consists of a label and a target.

We will write ι_i for $i \in \{1, \ldots, t\}$ for the injection into $\mathfrak{L}$ that corresponds to the ith component of the coproduct. They will also denote injections into $\mathfrak{L} \times X$: it should be clear from the context which of the two injection is meant.

As we will prove shortly, nominal transition systems are coalgebras in **Nom**, just as transition systems of sets are coalgebras over **Set**. Let L be the label functor, i.e., an endofunctor on $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$ of the form

$$L(-) = \coprod_{i=1}^{t} \mathbb{A}^{k_i} \times (-), \tag{9}$$

where $t \in \mathbb{N}_{>0}$ and $k_1, \ldots, k_t \in \mathbb{N}_{\geq 0}$. Note that, with t and all k_i fixed, $LX \cong \mathfrak{L} \times X$ for all nominal sets X.

Intuitively, L describes the labels of the transition relation, t is the number of terms or the number of 'kinds' of labels, and each k_i determines the number of names in those kinds of labels.

Example 4.3. Let X be a nominal set. To model input, output, and silent transitions, put

$$\mathfrak{L} = \mathbb{A}^2 + \mathbb{A}^2 + 1$$

We have $\mathfrak{L} \times X \cong (\mathbb{A}^2 \times X) + (\mathbb{A}^2 \times X) + X$, and so, $\mathfrak{L} \times X = LX$ for the endofunctor

$$L(-) = \mathbb{A}^2 \times (-) + \mathbb{A}^2 \times (-) + (-),$$

where we use $t = 3$, $k_1 = k_2 = 2$, and $k_3 = 0$.
In the π-calculus, it is customary to write the labels above the arrow. This can be done in the following way:

$$\begin{array}{ccccccc} x & \xrightarrow{a(b)} & y & :\Longleftrightarrow & x & \to & \iota_1(a,b,y) \\ x & \xrightarrow{a\langle b\rangle} & y & :\Longleftrightarrow & x & \to & \iota_2(a,b,y) \\ x & \xrightarrow{\tau} & y & :\Longleftrightarrow & x & \to & \iota_3(y) \end{array}$$

Although the nominal set of states X can contain name abstractions, none of the names in this equivariant transition system relation are bound in any formal way. Thus, as we will explain in more detail later, this is not how input transitions should be modeled, as the name b in a transition $x \xrightarrow{a(b)} y$ is supposed to be a binding name whose scope is y.

From now on, the endofunctor B is fixed as follows:

Definition 4.4. The endofunctor $B : \mathbf{Nom} \longrightarrow \mathbf{Nom}$ is given by $\mathcal{P}_{\mathrm{fs}} \circ L$. That is,

$$BX = \mathcal{P}_{\mathrm{fs}}(\mathfrak{L} \times X)$$

for every nominal set X. ■

The following lemma connects this functor to nominal transition systems:

Lemma 4.5. To give a B-coalgebra is to give a nominal transition system. ■

Proof. The equivariant relation $\rightarrow$ in **Nom** is an equivariant subset, and therefore, the characteristic function $\chi_\rightarrow$ is an equivariant function (Lemma 3.10). Hence, the relation $\rightarrow$ corresponds to a morphism $\chi_\rightarrow : X \times LX \longrightarrow \Omega$ in $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$, and through the natural bijection $\hom_{\mathbf{Nom}}(X \times LX, \Omega) \cong \hom_{\mathbf{Nom}}(X, (\Omega^{LX})_{\mathrm{fs}})$, this in turn corresponds to a morphism $X \longrightarrow \mathcal{P}_{\mathrm{fs}}LX$, which should be our coalgebraic structure. For a ground transition system $(X, \mathfrak{L}, \rightarrow)$ and a B-coalgebra $g : X \longrightarrow BX$, we explicitly correspond between the two by:

$$x \rightarrow \iota_i(a_1, \ldots, a_{k_i}, y) \iff \iota_i(a_1, \ldots, a_{k_i}, y) \in g(x)$$

☺

We would like our transition system to also support capture-avoiding substitutions, for which we can use the extra structure of nominal *renaming* sets. Recall that $U : \mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}} \longrightarrow \mathbf{Nom}$ is the forgetful functor.

Definition 4.6. A U-structured B-coalgebra, or B^U-coalgebra for short, is a pair (X, β), where X is an object of $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}}$, and $\beta : UX \longrightarrow BUX$ is a B-coalgebra structure (a morphism in **Nom**) on UX. ■

This way, our transition system of nominal sets can be accompanied by a capture-avoiding substitution structure on the set of states X. Indeed, the following corollary is straightforward:

Corollary 4.7. To give a B^U-coalgebra is to give a wide-open transition system of nominal sets. ■

4.2 Algebras in Nom: Syntax with Alpha-Equivalence

With transition systems of sets, in Section 2, we used a polynomial endofunctor S (using product and coproduct) to describe some syntax. Here we will do the same, but we will add the abstraction endofunctor to S. This will provide us to work with terms up to α-equivalence, which is how terms with locally instantiated names in process calculi are interpreted.

Compare the following definition with Definition 2.16:

Definition 4.8. [7] An *algebraic binding signature* is a set Σ of function symbols with, for each function symbol $\mathsf{f} \in \Sigma$, a name-arity $\sharp_{\mathbb{A}}(\mathsf{f}) \in \mathbb{N}$ and a term-arity $\sharp_X(\mathsf{f}) \in \mathbb{N}$. Furthermore, to each term of each function symbol is associated a number of binding names $\sharp_{\mathrm{bn}}(\mathsf{f})(i)$, where $i \in [1, \sharp_X(\mathsf{f})]$ indicates the ith term of the function symbol f. ■

Like in Section 2, for X a set, ΣX denotes the set of Σ-terms, where only one operator has been applied on elements of X, and $\Sigma^* X$ denotes the smallest set that contains X and is closed under application of operators of Σ.

For X a set, we write elements of ΣX as

$$\mathsf{f}(a_1, \ldots, a_{\sharp_{\mathbb{A}}\mathsf{f}}, \langle b_k^1 \rangle_{k \in [1, \sharp_{\mathrm{bn}}(\mathsf{f})(1)]} x_1, \ldots, \langle b_k^n \rangle_{k \in [1, \sharp_{\mathrm{bn}}(\mathsf{f})(n)]} x_n),$$

which is quite a laborious task because of the fact that every term is accompanied by a number of binding names. To avoid such tedious expressions, we will use shorthand notations $\widetilde{a} = a_1, \ldots, a_{\sharp_{\mathbb{A}}(\mathsf{f})}$ and $\widetilde{\langle b^i \rangle} = \langle b^i_k \rangle_{k \in [1, \sharp_{\mathsf{bn}}(\mathsf{f})(i)]}$, and leave the values of $\sharp_{\mathbb{A}}(\mathsf{f})$ and $\sharp_{\mathsf{bn}}(\mathsf{f})(i)$ for $i \in [1, \sharp_X(\mathsf{f})]$ implicit, whenever they should be clear from the context. The expression above will then be

$$\mathsf{f}(\widetilde{a}, \widetilde{\langle b^1 \rangle} x_1, \ldots, \widetilde{\langle b^n \rangle} x_n)$$

The reader should be very careful with the interpretation of the angular brackets $\langle \cdot \rangle$ here: they should *not* (yet) be interpreted as elements of name abstractions. At this point, it is just raw syntax.

Example 4.9. A portion of an algebraic binding signature for the π-calculus is $\Sigma = \{\mathsf{nil}, \mathsf{in}, \mathsf{out}, \mathsf{par}, \mathsf{sum}\}$, with

$\mathsf{f} \in \Sigma$	$\sharp_{\mathbb{A}}(\mathsf{f})$	$\sharp_X(\mathsf{f})$	$\sharp_{\mathsf{bn}}(\mathsf{f})(\ldots)$
nil	0	0	-
in	1	1	$\sharp_{\mathsf{bn}}(\mathsf{in})(1) = 1$
out	2	1	$\sharp_{\mathsf{bn}}(\mathsf{out})(1) = 0$
par	0	2	$\sharp_{\mathsf{bn}}(\mathsf{par})(1) = \sharp_{\mathsf{bn}}(\mathsf{par})(2) = 0$
sum	0	2	$\sharp_{\mathsf{bn}}(\mathsf{sum})(1) = \sharp_{\mathsf{bn}}(\mathsf{sum})(2) = 0$

If $P, Q \in X$ for some nominal set X, then $\mathsf{par}(P, Q)$ is often denoted $P \parallel Q$. Similarly, for some $a, b \in \mathbb{A}$, the term $\mathsf{in}(a, \langle b \rangle P)$ is better known as $a(b) \,.\, P$.

For a category $\mathbb{C}$ with finite product, coproduct, and abstraction, we can associate an endofunctor $S : \mathbb{C} \longrightarrow \mathbb{C}$ to an algebraic binding signature Σ, where

$$SY = \coprod_{\mathsf{f} \in \Sigma} \left(\mathbb{A}^{\sharp_{\mathbb{A}}(\mathsf{f})} \times \left[\mathbb{A}^{\sharp_{\mathsf{bn}}(\mathsf{f})(1)} \right] Y \times \cdots \times \left[\mathbb{A}^{\sharp_{\mathsf{bn}}(\mathsf{f})(\sharp_X(\mathsf{f}))} \right] Y \right) \tag{10}$$

Each term of the coproduct then represents an operator f of the signature, carrying $\sharp_{\mathbb{A}}(\mathsf{f})$ free names and $\sharp_X(\mathsf{f})$ terms with an amount of binding names equal to $\sharp_{\mathsf{bn}}(\mathsf{f})(i)$.

This initial algebra of closed Σ-terms is the algebra of interest:

Lemma 4.10. If S corresponds to a algebraic binding signature Σ as described in (10), then the set $S^*\emptyset = \Sigma^*\emptyset$ of closed Σ-terms carries an initial algebra structure for the syntax functor S. ■

Proof. Omitted. [7] ☺

Example 4.11. For $\mathbb{C} = \mathbf{Set}$, using the product $\mathbb{A}^k \times X$ for the abstraction endofunctor, the initial algebra for S contains all *raw* closed Σ-terms. A term $\langle b \rangle x$, e.g., being raw means that it is only syntax: the name b here is not in any formal way a binding name.

Example 4.12. For $\mathbb{C} = \mathbf{Nom}$, using the abstraction endofunctor in Definition 3.30, and for $\mathbb{C} = \mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$, using the lifting one from Lemma 3.31, the initial algebra contains all closed Σ-terms *up to α-equivalence*. This is exactly how we want to interpret a term such as $\langle b \rangle x$.
Moreover, in $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$ we can equip the terms up to α-equivalence with the renaming action of $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$, which is the usual capture-avoiding substitution used in process calculi.
If, for example, we want to perform the elementary renaming $[c/b]$ on a term $\mathsf{in}(a, \langle b \rangle(\mathsf{out}(a, b, \mathsf{nil})))$ in $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Pm}}$ (see Example 4.9, think of this term as $a(b) \;.\; a\langle b \rangle \;.\; \mathsf{nil}$), then Lemma 3.31 says that this can be done as follows:

$$[c/b] \cdot (\mathsf{in}(a, \langle b \rangle(\mathsf{out}(a, b, \mathsf{nil})))) = \mathsf{in}([c/b]a, \langle x \rangle([c/b] \cdot \mathsf{out}(a, x, \mathsf{nil})))$$

where $x \notin \{b, c\}$. Notice that we have *silently renamed* the binding name b to x. We are allowed to do this, since, by abstraction, $\langle b \rangle \mathsf{out}(a, b, \mathsf{nil}) = \langle x \rangle \mathsf{out}(a, x, \mathsf{nil})$.

4.3 Binding Names in NTS Labels

In the previous section, we provided a mathematical model that lets us interpret, e.g., a term $\langle b \rangle x$ as a term up to α-equivalence. For this, we used abstraction in **Nom**.

We point out now that, in a similar way, a name b of an input transition $x \xrightarrow{a(b)} y$ is usually interpreted as a binding name, whose scope is y. As we can see in Example 4.3, however, this is not in any way underpinned by abstraction, like was done with Σ-terms up to α-equivalence in the previous section.

In what follows we try to motivate our ambition to interpret b as a binding name in y, and we aim to make this formal in the nominal transition system in a more general way. Later, we will see how this relates to using abstraction (besides product and coproduct) in the label functor L defined in (9) for the coalgebra that corresponds to the NTS.

Consider the rule for communication in the early semantics [2] of the π-calculus (we also include the rule for early input):

$$\frac{P \xrightarrow{a\langle b \rangle} P' \qquad Q \xrightarrow{a(b)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\ \textsf{Comm-Early} \tag{11}$$

In this rule, the name that is sent by P needs to be exactly the same as the name received by Q. But suppose $Q \triangleq a(c) \;.\; a\langle c \rangle \;.\; \mathsf{nil}$. Many process calculi will have a rule specifying that then $Q \xrightarrow{a(c)} a\langle c \rangle \;.\; \mathsf{nil}$. But then, how can Q receive a name $b \neq c$ through a from another process P? The bound name c in the transition needs to be replaced by b first.

[2]The word *'early'* here refers to the moment that substitution of the input name takes place, and not to the time in which the semantics was conceived. Early and late substitution in communication rules has been studied and is sometimes referred to as *internal* and *external* mobility. [20]

One could argue now that Q, as a term, is α-equivalent to $a(b)\,.\,a\langle b\rangle\,.\,\mathsf{nil}$, and that there is no issue here. Note, however, that we do not want to presuppose a syntax structure on the states: they only serve for illustrative purposes here. We wish to formalize the concept of binding names solely in terms of the nominal transition system itself (and later, the coalgebra).

Moreover, even when considering this particular syntax and realizing that we have $Q \approx_\alpha a(b)\,.\,a\langle b\rangle\,.\,\mathsf{nil}$, it still seems fruitful, necessary even, to require that a transition is independent on the choice of representative from the α-equivalence class of the source.

Our goal is now to formulate this requirement for an NTS. For this, we have to know which names in the label are supposed to be binding. We therefore update our definition of a nominal transition system as follows:

Definition 4.1 (*Updated*)**.** A *nominal transition system* is a quadruple $(X, \mathfrak{L}, \rightarrow, \textsc{bn})$ where the carrier X is a nominal set, $\mathfrak{L}$ is a nominal set of labels, the transition relation $\rightarrow \subseteq X \times (\mathfrak{L} \times X)$ is equivariant, and $\textsc{bn} : \mathfrak{L} \longrightarrow \mathcal{P}_\omega(\mathbb{A})$ is an equivariant function giving a subset of names such that $\textsc{bn}(l) \subseteq \operatorname{supp}_{\mathfrak{L}}(l)$ for all $l \in \mathfrak{L}$.

The quadruple is called a *nominal renaming transition system* if X is a nominal renaming set. ■

The function $\textsc{bn}$ does not change anything to the interpretation of the set of names that it gives, but it enables us to formulate a property that establishes some names as being binding in the target of a transition. This can be done in two possible ways:

Definition 4.13. We say that a nominal transition system $(X, \mathfrak{L}, \rightarrow, \textsc{bn})$ satisfies *strong alpha-conversion of residuals* (SACR) if for all transitions $x \xrightarrow{l} y$ and for all $b \in \textsc{bn}(l)$:

$$\forall a \in \mathbb{A}\,.\, a \# (l, y) \implies x \xrightarrow{(a\ b)\cdot l} (a\ b)\cdot y.$$

It satisfies *weak alpha-conversion of residuals* (WACR) if for all transitions $x \xrightarrow{l} y$ and for all $b \in \textsc{bn}(l)$:

$$\forall a \in \mathbb{A}\,.\, a \# (l, y, x) \implies x \xrightarrow{(a\ b)\cdot l} (a\ b)\cdot y.$$

■

The difference between strong and weak alpha-conversion is the freshness condition in the assumption of the implication. For both versions we need $a \# l$, for strong conversion $a \# y$ suffices, whereas for weak conversion, $a \# x$ is required in addition.

It is clear that any NTS that satisfies SACR also satisfies WACR.

Note that the authors of [1, 6, 24] refer to strong alpha-conversion of residuals as just alpha-conversion of residuals, but we wish to explicitly distinguish between strong and weak alpha-conversion for reasons that will become clear later.

Also, the property of (strong) alpha-conversion of residuals is included in the definition of nominal transition systems in [1, 6, 24].

Example 4.14. Recall the endofunctor B from Definition 4.4. Any B-coalgebra is a nominal transition system that satisfies SACR if we define $\textsc{bn}$ to give the empty set for all labels by Lemma 4.5, as the property of SACR then quantifies over the emptyset $\textsc{bn}(l)$ for every transition $x \xrightarrow{l} y$, and is therefore vacuously satisfied.

Example 4.15. The nominal transition system of Example 4.3 needs to be extended with a function $\textsc{bn}$. For a transition $x \xrightarrow{l} y$, we let $\textsc{bn} : l \mapsto \{b\}$ if and only if $l = \iota_1(a,b)$ and $\textsc{bn}(l) = \emptyset$ otherwise. (Recall that we made the first injection ι_1 correspond to input.)

Then, if we have, e.g., a transition $x(b) \,.\, y\langle b\rangle \,.\, \mathsf{nil} \xrightarrow{x(b)} y\langle b\rangle \,.\, \mathsf{nil}$, if the NTS satisfies SACR or just WACR, we also have $x(b) \,.\, y\langle b\rangle \,.\, \mathsf{nil} \xrightarrow{x(a)} y\langle a\rangle \,.\, \mathsf{nil}$ for any name $a \notin \{x,y\}$ (x occurs in the label, and y occurs in the target, as well as the source).

Now consider $P \triangleq x(b) \,.\, y\langle b\rangle \,.\, \mathsf{nil} + z\langle z\rangle \,.\, \mathsf{nil}$ with a transition $P \xrightarrow{x(b)} y\langle b\rangle \,.\, \mathsf{nil}$. If the system satisfies SACR, then we also have $P \xrightarrow{a(z)} y\langle z\rangle \,.\, \mathsf{nil}$, since z is fresh in the label and in the target $y\langle b\rangle \,.\, \mathsf{nil}$. However, for WACR, this is not necessarily the case, since z is not fresh in the source P.

Since Definition 4.1 has been updated, Lemma 4.5 and Corollary 4.7 are no longer valid. In order to reclaim their validity, we need to add an important assumption on the structure of $\textsc{bn}$. For this, recall from Assummption 4.2 that $\mathfrak{L} = \coprod_{i=1}^{t} \mathbb{A}^{k_i}$.

Assumption 4.16. For each coproduct term, the number of binding names is the same for all labels therein:

$$\forall \iota_i(a_1,\ldots,a_{k_i}), \iota_j(a'_1,\ldots,a'_{k_j}) \in \mathfrak{L} \,.\, i = j \implies |\textsc{bn}(\iota_i(a_1,\ldots,a_{k_i}))| = |\textsc{bn}(\iota_j(a'_1,\ldots,a'_{k_j}))|.$$

Furthermore, the names are ordered in such a way that all binding names occur last:

$$\forall \iota_i(a_1,\ldots,a_{k_i}) \in \mathfrak{L} \,.\, m = |\textsc{bn}(\iota_i(a_1,\ldots,a_{k_i}))| \implies \textsc{bn}(\iota_i(a_1,\ldots,a_{k_i})) = \{a_{k_i-m+1},\ldots,a_{k_i}\}$$

■

This assumption is not unworkable when we consider the π-calculus, for example, where every input label has exactly one binding name. Moreover, assuming the binding names occur last in the label (or at least have a fixed position) is also in line with common practice in process calculi.

Finally, to reestablish the correspondence of B-coalgebras with nominal transition systems, we have to equip a B-coalgebra with a function $\mathfrak{b} : \{1,\ldots,t\} \longrightarrow \mathbb{N}$ indicating the number of binding names for each coproduct name.

We reiterate the correspondence between B-coalgebas and nominal transition systems:

Lemma 4.4 (*Updated*)**.** To give a B-coalgebra with a function $\mathfrak{b} : \{1,\ldots,t\} \longrightarrow \mathbb{N}$ is to give a nominal transition system. ■

Proof. The only new thing we have to show is correspondence between the functions $\mathfrak{b}$ and BN. By Assummption 4.16, the number of binding names for the ith coproduct term is fixed. Call this number m_i. Thus, we can correspond between $\mathfrak{b}$ and BN by

$$\mathfrak{b} : i \mapsto m_i \qquad \Longleftrightarrow \qquad \text{BN} : \iota_i(a_1, \ldots, a_{k_i}) \mapsto \{a_{k_i - m_i + 1}, \ldots, a_{k_i}\}$$

☺

Corollary 4.7 is in the same way still valid if a B^U-coalgebra is equipped with a function $\mathfrak{b} : \{1, \ldots, t\} \longrightarrow \mathbb{N}$.

4.4 Coalgebras with Name Abstraction

We have seen that nominal transition systems are coalgebras equipped with some function indicating the number of binding names for each type of label. However, the function BN in this correspondence is not used in any way to require that the names are binding. This only happens when we say that the NTS satisfies strong or weak α-conversion of residuals.

In this section we look at coalgebras where the label functor L uses the abstraction endofunctor, and we investigate the manner in which this relates to the properties SACR and WACR. Will abstraction in the label functor be the way to model name binding in the nominal transition system?

With the function $\mathfrak{b}$ indicating the number of binding names for a B-coalgebra, we let

$$\widetilde{B} = \mathcal{P}_{\text{fs}} \widetilde{L} \qquad \text{with} \qquad \widetilde{L} Y = \coprod_{i=1}^{t} \mathbb{A}^{k_i - \mathfrak{b}(i)} \times \left[\mathbb{A}^{\mathfrak{b}(i)}\right] Y$$

There should be enough names to bind in the i-th term, so we assume $\mathfrak{b}(i) \leq k_i$ for all $i \in \{1, \ldots, t\}$.

We are interested in the relation between this kind of coalgebra with nominal transition systems. Since B-coalgebras are nominal transition systems by Lemma 4.5, it is fruitful to investigate to relation between B-coalgebras and $\widetilde{B}$-coalgebras, where we assume that t (the number of coproduct terms) and k_i for all $i \in \{1, \ldots, t\}$ (the number of names involved in every coproduct term) are the same for L in B and $\widetilde{L}$ in $\widetilde{B}$.

We will claim in Theorem 4.21 that $\widetilde{B}$-coalgebras are B-coalgebras satisfying the following property:

Rule ($\star$). Let (X, g) be a B-coalgebra. We say that g satisfies Rule ($\star$) if

$$(\star) \quad \forall i \in \{1, \ldots, t\}. \quad \forall a_1, \ldots, a_{k_i} \in \mathbb{A}. \quad \forall x, y \in X.$$
$$\iota_{L,i}(a_1, \ldots, a_{k_i}, y) \in g(x) \implies \{a_{k_i + 1 - \mathfrak{b}(i)}, \ldots, a_{k_i}\} \# \{x, a_1, \ldots, a_{k_i - \mathfrak{b}(i)}\}$$

■

That is, the binding names in the label of any transition are fresh in the source and distinct from the other names in the label.

Example 4.17. In Example 4.3 we gave an NTS, which was a B-coalgebra with

$$L(-) = \mathbb{A}^2 \times (-) + \mathbb{A}^2 \times (-) + (-).$$

Here, $t = 3$, $k_1 = k_2 = 2$, and $k_3 = 0$. Recall that the first coproduct term represented input, so let $\mathfrak{b} : \{1,2,3\} \longrightarrow \mathbb{N}$ be such that $\mathfrak{b}(1) = 1$ (one binding input name) and $\mathfrak{b}(2) = \mathfrak{b}(3) = 0$.
If this B-coalgebra satisfies Rule ($\star$), then the freshness relations are vacuously satisfied for $i \in \{2,3\}$. For $i = 1$, the rule says that if $\iota_{L,1}(a,b,y) \in g(x) \implies b \# \{a,x\}$.
In the nominal transition system, this means that $b \# \{a,x\}$ whenever $x \xrightarrow{a(b)} y$.

By merit of Rule ($\star$), the entire equivalence class (restricted to suitably fresh names) is part of the set of transition targets for x if only one member is:

Lemma 4.18. Let (X,g) be a B-coalgebra satisfying Rule ($\star$) and $\iota_{L,i}(a_1,\dots,a_{k_i},y) \in g(x)$ for some $(a_1,\dots,a_{k_i},y) \in \mathbb{A}^{k_i} \times X$ and $x \in X$. To relieve us from some heavy notation, write $\widetilde{a} = a_1,\dots,a_{k_i-\mathfrak{b}(i)}$ for the free names, $\widetilde{\boldsymbol{a}} = a_{k_i+1-\mathfrak{b}(i)},\dots,a_{k_i}$ for the bound names, and $\widetilde{b} = b_1,\dots,b_{\mathfrak{b}(i)}$. We then have

$$\forall(\widetilde{b},y') \in \mathbb{A}^{\mathfrak{b}(i)} \times X \,.\, (\widetilde{b},y') \approx_\alpha (\widetilde{\boldsymbol{a}},y) \wedge \{\widetilde{b}\} \# \{x,\widetilde{a}\} \implies \iota_{L,i}(\widetilde{a},\widetilde{b},y') \in g(x)$$

■

Proof. Let $(\widetilde{b},y') \in \mathbb{A}^{\mathfrak{b}(i)} \times X$ where $(\widetilde{b},y') \approx_\alpha (\widetilde{\boldsymbol{a}},y)$ and $\{\widetilde{b}\} \# \{x,\widetilde{a}\}$. By Rule ($\star$), we have $\{\widetilde{\boldsymbol{a}}\} \# \{x,\widetilde{a}\}$.

By definition of α-equivalence, there is π such that $\pi \cdot (\widetilde{\boldsymbol{a}},y) = (\widetilde{b},y')$ with $\mathrm{supp}(\pi) \subseteq \{\widetilde{b},\widetilde{\boldsymbol{a}}\}$. Since $\{\widetilde{b},\widetilde{\boldsymbol{a}}\} \# \{\widetilde{a}\}$, then, we have $\pi \cdot \widetilde{a} = \widetilde{a}$.

Since $\{\widetilde{b}\} \# x$ and $\{\widetilde{\boldsymbol{a}}\} \# x$ and g is equivariant, we obtain $\{\widetilde{b},\widetilde{\boldsymbol{a}}\} \# g(x)$ by Lemma 3.27. But then $\pi \# g(x)$, because $\mathrm{supp}(\pi) \subseteq \{\widetilde{b},\widetilde{\boldsymbol{a}}\}$. From this, we know that

$$\forall a \in \mathrm{supp}(g(x)) \,.\, \pi a = a,$$

and with this we can apply Lemma 3.24 to obtain

$$\pi \cdot \iota_{L,i}(\widetilde{a},\widetilde{\boldsymbol{a}},y) \in g(x).$$

The result follows then by realizing that $\pi \cdot \iota_{L,i}(\widetilde{a},\widetilde{\boldsymbol{a}},y) = \iota_{L,i}(\widetilde{a},\widetilde{b},y')$. ☺

Example 4.19. Consider the same NTS (B-coalgebra) that we used in Example 4.17 and assume that the B-coalgebra satisfies Rule ($\star$). Let $x \xrightarrow{a(b)} y$ be a transition and assume $c \# \{a,y,x\}$. Note that now $(b,y) \approx_\alpha (c,(b\ c)\cdot y)$ by Lemma 3.29.
Now Lemma 4.18 says that we also have the transition $x \xrightarrow{a(c)} (b\ c)\cdot y$.
This is because the transition relation is equivariant, so $(b\ c)\cdot x \xrightarrow{a(c)} (b\ c)\cdot y$ and $(b\ c)\cdot x = x$, since both b and c are fresh in x. (The proof of the lemma is like this, but it uses the coalgebra structure instead of the transition relation.)
Note that this shows that the NTS corresponding to this B-coalgebra satisfies weak alpha-conversion of residuals.

The property stated in Lemma 4.18 can be reformulated for the corresponding nominal transition system. This way, we easily arrive at the following corollary:

Corollary 4.20. A nominal transition system corresponding to a B-coalgebra equipped with some function $\mathfrak{b} : \{1,\ldots,t\} \longrightarrow \mathbb{N}$ that satisfies Rule $(\star)$ has weak alpha-conversion of residuals. ■

Our main result is now that coalgebras with abstraction are nominal transition systems satisfying weak alpha-conversion of residuals:

Theorem 4.21 (*Main result*)**.** There is a bijective correspondence between $\widetilde{B}$-coalgebras and B-coalgebras with some function $\mathfrak{b} : \{1,\ldots,t\} \longrightarrow \mathbb{N}$ satisfying Rule $(\star)$. ■

Corollary 4.22. $\widetilde{B}$-coalgebras are nominal transition systems satisfying weak alpha-conversion of residuals. ■

Proof (of Theorem 4.21). We correspond between B-coalgebras that satisfy Rule $(\star)$ and $\widetilde{B}$-coalgebras with:

$$\begin{tikzcd} B\text{-}\mathbf{Coalg} \arrow[r, shift left, "\phi"] & \widetilde{B}\text{-}\mathbf{Coalg} \arrow[l, shift left, "\psi"] \end{tikzcd}$$

$$\begin{array}{rcl} (X, g) & \mapsto & (\widetilde{g} : x \mapsto V_x) \\ (g : x \mapsto W_x) & \leftarrow\!\shortmid & (X, \widetilde{g}) \end{array}$$

where

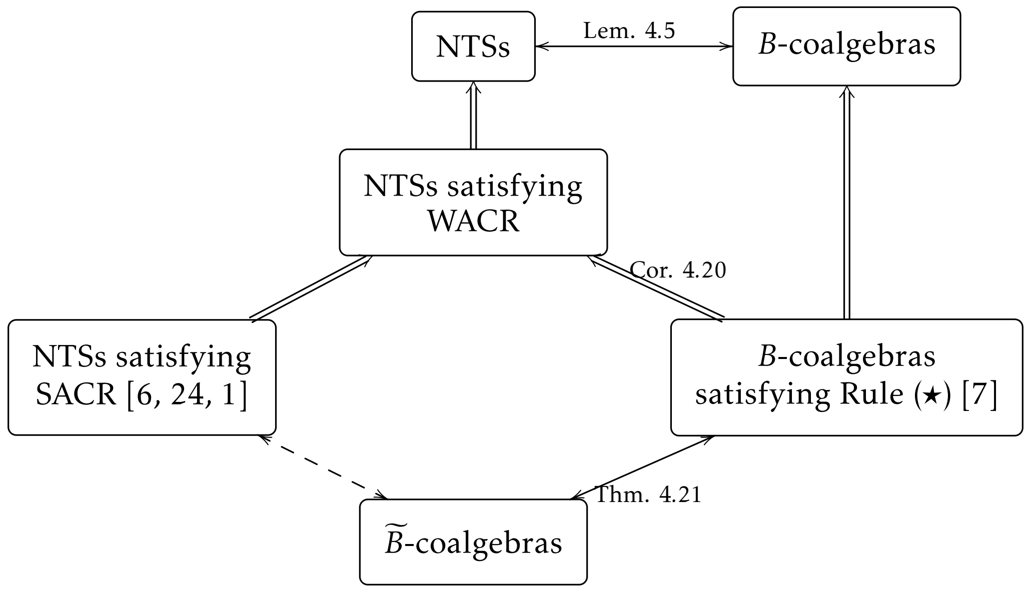
$$V_x = \coprod_{i=1}^{t} \left\{ \iota_{\widetilde{L},i}(a_1,\ldots,a_{k_i-\mathfrak{b}(i)},\langle a_{k_i-\mathfrak{b}(i)+1},\ldots,a_{k_i}\rangle y) \;\middle|\; \iota_{L,i}(a_1,\ldots,a_{k_i},y) \in g(x) \right\}$$

$$\begin{aligned} W_x = \coprod_{i=1}^{t} \Big\{ \iota_{L,i}(a_1,\ldots,a_{k_i-\mathfrak{b}(i)},b_1,\ldots,b_{\mathfrak{b}(i)},y') \;\Big|\; & \iota_{\widetilde{L},i}(a_1,\ldots,a_{k_i-\mathfrak{b}(i)},\langle a_{k_i-\mathfrak{b}(i)+1},\ldots,a_{k_i}\rangle y) \in \widetilde{g}(x), \\ & (b_1,\ldots,b_{\mathfrak{b}(i)},y') \approx_\alpha (a_{k_i-\mathfrak{b}(i)+1},\ldots,a_{k_i},y), \quad \{b_1,\ldots,b_{\mathfrak{b}(i)}\} \# \{x,a_1,\ldots,a_{k_i-\mathfrak{b}(i)}\} \Big\} \end{aligned}$$

The proof can be found in Appendix B.4. We use Lemma 4.18 to show that $\psi \circ \phi$ gives back the same B-coalgebra that satisfies Rule $(\star)$ by construction. ☺

5 Conclusion and Future Work

We summarize the theory we described in Section 4 as follows:


NTSs
Lem. 4.5
B-coalgebras
NTSs satisfying WACR
Cor. 4.20
NTSs satisfying SACR [6, 24, 1]
B-coalgebras satisfying Rule (★) [7]
Thm. 4.21
$\widetilde{B}$-coalgebras

$\Longrightarrow$ ... is a subclass of ...

$\longleftrightarrow$ ... corresponds with ...

$\leftarrow - \rightarrow$ conjectured correspondence

As we have seen in Section 4.2, one can straightforwardly set up a syntax for a process calculus as a nominal renaming set that supports name-by-name substitution. The challenge in constructing nominal transition systems is rather to make sure that a rule specification for the behaviour induces an equivariant transition relation. Both formats in [1] and [7] succeed in this regard, but they differ in the way they treat binding names in the label. It would be perfect if we can make sure that, through the correspondence of nominal transition systems with coalgebras, name binding in labels is exactly name abstraction in the endofunctor for the labels of the coalgebra.

Strong Alpha-Conversion of Residuals The format in [1] by Aceto et al. ensures that the transition system relation is equivariant and that it satisfies ACR (here SACR). They call it the *ACR format* and it requires one to recursively define a *partial strict stratification* function $S^*\emptyset \longrightarrow \mathbb{N}$ on the syntax of processes. The format guarantees that a specification $\mathcal{R}$ induces an equivariant transition relation by defining a permutation action on rules, and requiring that for every rule Ru in $\mathcal{R}$ and every $a, b \in \mathbb{A}$, the rule $(a\ b) \cdot \mathsf{Ru}$ is also in $\mathcal{R}$.

Furthermore, the format has three constraints to ensure ACR, which can be summarized as follows: there is a substitution γ whose domain only contains terms from the conclusion source that do not occur anywhere else, such that

(i) after any substitution, fresh names in the conclusion target should also be fresh in the premise targets;

(ii) after any substitution, names that are fresh in the conclusion target and in all premise sources are also fresh in the conclusion source (after applying γ); and

(iii) after any substitution, binding names in premise labels are fresh for the conclusion source (after applying γ).

Using induction, the first two constraints guarantee that, for any transition, a name that is fresh in the target is also fresh in the source (after applying γ). The third constraint ensures that, by induction on the strict stratification function, any bound name indicated by BN is fresh in the conclusion source (after applying γ).

Then, after showing that γ does not change the fact that there exists a proof tree for the rule (since it only affects irrelevant parts in the conclusion source), with these constraints in place, (strong) ACR follows.

Note that in this thesis, we made significant assumptions on the structure of the label set $\mathfrak{L}$: their set of labels remains very generic, whereas we, by Assummption 4.2, only consider a particular structure. This means in particular that we rule out higher order process calculi, in which processes can be passed around through channels.

Nominal GSOS Format In [7], Staton & Fiore prove that a specification in their format can be transformed in an abstract rule

$$\rho : \Sigma(U \times BU) \Rightarrow BU\Sigma^* : \mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}} \to \mathbf{Nom}.$$

This abstract rule will provide an initial transition system that satisfies congruence of bisimilarity. The B-coalgebra that is part of this bialgebra supports name abstraction, and they claim that this name abstraction in the coalgebra is equivalent to saying that binding names are fresh in the source for any transition.

In this thesis, we have proven this claim and made this correspondence explicit by Theorem 4.21.

To state the constraints of their format requires quite a bit of detailed work. Their most significant assumption in relation to this thesis is that they assume that any bound name in the conclusion label is fresh for the conclusion source. This instantly entails that the coalgebraic structure of the intended transition system satisfies Rule ($\star$).

We have showed that B-coalgebras satisfying Rule ($\star$) are nominal transition systems satisfying WACR (Corollary 4.20), but we have not shown the converse: it is not clear wether weak alpha-conversion of residuals implies Rule ($\star$).

Future Work The property of strong alpha-conversion states that if one residual is the target of a transition from some state x, then the whole equivalence of the residual must be a target of a transition from x. This easily leads one to conjecture a correspondence between nominal transition systems (under Assumptions 4.2 and 4.16) and $\widetilde{B}$-coalgebras. Although in [1], terms of the system are related to nominal sets, their theory has not yet been related to coalgebras. Combining the format of Staton & Fiore [7] with some aspects of Aceto's format [1], we might obtain NTSs that satisfy SACR that are also $\widetilde{B}$-coalgebras.

Rule ($\star$) seems to be too strict to encode in the nominal GSOS format [7]. It rules out a transition $x(x)\,.\,\mathsf{nil} + z\langle z\rangle\,.\,\mathsf{nil} \xrightarrow{x(z)} \mathsf{nil}$, even though this seems like a perfectly legitimate transition. It is worthwile to investigate how to potentially alleviate some of the heavy restrictions of this format.

Although we were unable to find any references on bisimilarity in **Nom**, the idea that the greatest bisimulation exists for coalgebras in **Nom**, i.e., that the filtered colimit of bisimulations is again a bisimulation, is assumed throughout the literature. This is something worth investigating.

It is known that M-**Set** and **Nom** are Boolean topoi (cartesian closed categories with a subobject classifier). Still, internal constructions such as the exponent and the power object are not very well-documented. These can, however, be of great aid in defining, e.g., a lifting from **Nom** to $\mathbf{Nom}_{\mathbb{A}}^{\mathrm{Rn}}$ for a generalized abstraction endofunctor [5] ($[X]\,Y$ rather than $\left[\mathbb{A}^k\right]X$, as we did in Lemma 3.31). This will also be fruitful in potentially letting go of assumptions on the transition labels for nominal transition systems. Some work on this has already been done in [8] and [22] (unpublished).

Acknowledgements

I thank Jurriaan Rot for his ideas, his patience and guidance throughout this project. I am also thankful for the thorough feedback provided by Thorsten Wißmann. Finally, I am very grateful to Harold Simmons, whose unpublished notes have been of great benefit to me.

Appendices

A Category Theory

A.1 Categories, Functors and Natural Transformations

A category $\mathbb{C}$ consists of

(i) a class of *objects*;

(ii) for each two objects X and Y of $\mathbb{C}$ a set $\hom_{\mathbb{C}}(X,Y)$ of *morphisms* or arrows $a : X \longrightarrow Y$, whose *domain* is X and *codomain* is Y, sometimes denoted $X \xrightarrow{a} Y$; and

(iii) a binary operation on arrows called *composition* that gives an arrow $c : X \longrightarrow Z$ for each $X \xrightarrow{a} Y \xrightarrow{b} Z$, written $c = b \circ a$, or simply $c = ba$,

such that the following hold:

(1) for every object X of $\mathbb{C}$, there is an *identity morphism* $\mathrm{id}_X : X \longrightarrow X$ that satisfies $\mathrm{id}_X \circ a = a$ for all $a : W \longrightarrow X$ and $b \circ \mathrm{id}_X = b$ for all $b : X \longrightarrow Y$, and

(2) composition of morphisms is *associative*, meaning that we have $(c \circ b) \circ a = c \circ (b \circ a)$ for all $W \xrightarrow{a} X \xrightarrow{b} Y \xrightarrow{c} Z$.

A category is *small* if its class of objects is a set. Some authors let $\hom_{\mathbb{C}}(X,Y)$ be a class and adopt the above definition, where $\hom_{\mathbb{C}}(X,Y)$ is a set, for *locally small* categories. We are assuming all our categories are locally small.

A *functor* F from a category $\mathbb{C}$ to a category $\mathbb{D}$ associates to every object X and morphism a of $\mathbb{C}$ an object FX or morphism Fa respectively in $\mathbb{D}$, preserving structure in the following three ways:

(i) Every morphism $c : X \longrightarrow Y$ of $\mathbb{C}$ is associated to a morphism $Fc : FX \longrightarrow FY$ of $\mathbb{D}$.

(ii) Identity morphisms are preserved, meaning $F(\mathrm{id}_X) = \mathrm{id}_{FX}$ for all objects X of $\mathbb{C}$.

(iii) Composition of morphisms is preserved, meaning that, for all $X \xrightarrow{a} Y \xrightarrow{b} Z$ in $\mathbb{C}$, we have $F(ba) = F(b)F(a)$.

An *endofunctor* is a functor from a category $\mathbb{C}$ to the same category $\mathbb{C}$, e.g., $\mathrm{Id}_{\mathbb{C}} : \mathbb{C} \longrightarrow \mathbb{C}$.

A natural transformation λ from a functor $F : \mathbb{C} \longrightarrow \mathbb{D}$ to a functor $G : \mathbb{C} \longrightarrow \mathbb{D}$, written $\lambda : F \Rightarrow G$, is a family (class) of morphisms

$$\{\lambda_X : FX \longrightarrow GX\}_{X \in \mathbb{C}}$$

in $\mathbb{D}$ such that for every morphism $a : X \longrightarrow Y$ in $\mathbb{C}$, the *naturality* condition holds:

$$\lambda_Y \circ Fa = Ga \circ \lambda_X$$

Since the naturality condition is an equality of composed morphisms, one can also formulate this equality by stating that the following diagram of arrows *commutes*, meaning that the choice of path is irrelevant:

$$\begin{tikzcd}
FX \arrow[r, "\lambda_X"] \arrow[d, "Ff"] & GX \arrow[d, "Gf"] \\
FY \arrow[r, "\lambda_Y"] & GY
\end{tikzcd}$$

If $\mathbb{D}$ is a small category, the class of all functors $F : \mathbb{D} \longrightarrow \mathbb{C}$, denoted $[\mathbb{D}, \mathbb{C}]$, forms a category, where the morphisms are natural transformations $\lambda : F \Rightarrow G$ between functors $F, G : \mathbb{D} \longrightarrow \mathbb{C}$.

A.2 Products, Subobjects and Relations

An *initial* object I of a category $\mathbb{C}$ is an object that has exactly one outgoing arrow to each object X of $\mathbb{C}$. Dually, *final* or *terminal* objects have unique *incoming* arrows from every object of $\mathbb{C}$.

Example A.1. In **Set**, a final object is a set $\{*\}$. This codomain forces all functions to send everything to the single element. The initial object of **Set** is the empty set $\emptyset$, from which every unique outgoing arrow is the empty function.

Since there is an obvious isomorphism between any existing two initial objects, we often refer to initial objects up to isomorphism, and call it *the* initial object, as we do when referring to final objects as *the* final object.

A *span* is a pair of morphisms $a : C \longrightarrow A$ and $b : C \longrightarrow B$, for which we will write $A \xleftarrow{a} C \xrightarrow{b} B$. Dually, we write $A \xrightarrow{a} C \xleftarrow{b} B$ for *cospans*.

Definition A.2. A span $X \xleftarrow{\pi_1} P \xrightarrow{\pi_2} Y$ in $\mathbb{C}$ is a *pullback* of a cospan $X \xrightarrow{f} Z \xleftarrow{g} Y$ if $f \circ \pi_1 = g \circ \pi_2$ and it satisfies the following universal property: for every span $X \xleftarrow{\rho_1} W \xrightarrow{\rho_2} Y$ such that $f \circ \rho_1 = g \circ \rho_2$, there is a unique morphism $q : W \longrightarrow P$ such that $\rho_1 = \pi_1 \circ q$ and $\rho_2 = \pi_2 \circ q$, i.e., the following diagram commutes:

$$\begin{tikzcd}
W \arrow[r, dashed, "!q"'] \arrow[rr, bend left, "\rho_2"] \arrow[dr, bend right, "\rho_1"'] & P \arrow[r, "\pi_2"'] \arrow[d, "\pi_1"] & Y \arrow[d, "g"] \\
 & X \arrow[r, "f"] & Z
\end{tikzcd}$$

A *weak pullback* is the same as a pullback, but the $q : W \longrightarrow P$ need not be unique. ■

We say that a functor preserves pullback if we obtain a pullback square after applying the functor on a pullback square. It preserves weak pullback if we obtain a weak pullback after applying it to a (weak) pullback square.

Definition A.3. A *product* $X \times Y$ of two objects X and Y of $\mathbb{C}$ is the pullback over the unique cospan from X and Y to the terminal object T:

$$\begin{tikzcd} X \times Y \arrow[r, "\pi_2"] \arrow[d, "\pi_1"] & Y \arrow[d, dashed, "!z_Y"] \\ X \arrow[r, dashed, "!z_X"] & T \end{tikzcd}$$

■

Again, if products exist in a category, then they are unique up to isomorphism, and we refer to a product as *the* product.

In a completely dual way, one can define the coproduct '+' as the *pushout* of the unique span $X \xleftarrow{\alpha_X} 0 \xrightarrow{\alpha_Y} Y$, which will result in a unique (up to isomorphism) cospan $X \xrightarrow{\iota_1} X+Y \xleftarrow{\iota_2} Y$.

Example A.4. In **Set**, the representative of the isomorphism class of products is given by the Cartesian product $X \times Y$ of two sets X and Y containing all pairs (x, y) where x and y range over X and Y respectively.
A convenient representative of the isomorphism class of pushouts is the *disjoint union* of sets, along with the two obvious injections of set inclusion.

Relations $R \subseteq X \times Y$ in **Set** can be thought of as an injection '$\subset$' from R to $X \times Y$. The categorical construction for this uses the notion of a *monomorphism*.

Definition A.5. A morphism $f : Y \longrightarrow Z$ is called a *monomorphism* (a *mono*, or *monic*), if for any pair of morphisms $X \overset{g}{\underset{h}{\rightrightarrows}} Y$ the implication $fg = fh \implies g = h$ holds. ■

We write $f : Y \hookrightarrow Z$ to indicate that f is monic.

Example A.6. In **Set**, subset injections $i : X \xrightarrow{\subset} Y$ are monic. Indeed, if for any two functions $f, g : W \longrightarrow X$, we have $i(f(w)) = i(g(w))$ for all $w \in W$, then, by injectivity, $f(w) = g(w)$ for all $w \in W$, and so $f = g$.

Lemma A.7. Let $Y \overset{f}{\hookrightarrow} Z$ and $X \xrightarrow{g} Y$ be morphisms. If $f \circ g$ is a monomorphism, then so is g. ■

Proof. For $W \overset{h_1}{\underset{h_2}{\rightrightarrows}} X$, assume $gh_1 = gh_2$. Then post-compose with f to get $fgh_1 = fgh_2$. Now, since fg is monic, we get $h_1 = h_2$, so indeed, g is monic. ☺

Let Mono(X) denote the class of monomorphisms with codomain X. The class can be equipped with a preorder $\leq$ as follows: for $V \xrightarrow{f} X \xleftarrow{g} W$, $f \leq g$ if and only if there is a morphism $h : V \longrightarrow W$ such that $gh = f$. The ordering is reflexive by existence of identity morphisms, and can be shown to be transitive by composing the existing morphisms h.

This preorder gives rise to an equivalence by letting $f \backsimeq g$ if and only if $f \leq g$ and $g \leq f$. Note that the domains of f and g are isomorphic if $f \backsimeq g$, since then we have $f = gh_1$ and $g = fh_2$, so that $f = f(h_2h_1)$ and $g = g(h_1h_2)$. Since f and g are monic, h_2h_1 and h_1h_2 must be identities.

Finally, to generalize the notion of *subset*, a *subobject* D of an object C is an equivalence class of $\backsimeq$ of monomorphisms with codomain C. When the monomorphism is clear from the context, we refer to the object as the subobject, and we often silently pick a convenient representative of the equivalence class.

Example A.8. In **Set**, subobject monomorphisms are injections into a set X and two injections are equivalent if their image is the same. A convenient representative from this equivalence class is the subset that *is* this image in X.

Assuming $\mathbb{C}$ has a terminal object 1, an object Ω, along with a morphism $t : 1 \longrightarrow \Omega$, is a *subobject classifier* if, for any subobject $D \xrightarrow{i} C$, there is a unique morphism $\chi_C : C \longrightarrow \Omega$ such that the following square is a pullback diagram:

$$\begin{tikzcd}
D \arrow[r] \arrow[d, "i"] & 1 \arrow[d, "t"] \\
C \arrow[r, dashed, "!\chi_C"] & \Omega
\end{tikzcd}$$

Example A.9. In **Set**, the subobject classifier is the two-element set $\Omega = \{\bot, \top\}$ with the morphism t picking a truth value: $t : * \mapsto \top$. Indeed, if the square above commutes, for $x \in C$, we have $x \in D$ if $\chi_C(x) = \top$ and we have $x \notin D$ if $\chi_C(x) = 0$. The universal property of D being a pullback here means that for any other morphism $V \xrightarrow{f} C$ such that $\chi_C \circ f = t \circ !_V$ (the morphism $!_V : V \longrightarrow 1$ is unique), f factors uniquely through D, which is indeed the situation with subsets.

Set theoretical relations are just subsets of the Cartesian product, which explains the following definition:

Definition A.10. Assume that $\mathbb{C}$ has products, and let X and Y be objects of $\mathbb{C}$. A *relation* R between X and Y is a subobject of $X \times Y$. Equivalently, the pairing $(\pi_1, \pi_2) : R \longrightarrow X \times Y$ of π_1 and π_2 in the span $X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$ is a monomorphism. ■

When the projections are clear from context, we refer to R itself as the relation.

Example A.11. In **Set**, take $X = \{a, b\}$ and $Y = \{c, d\}$. Then $X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$ with $R = \{(a, c), (a, d)\}$ is a relation in the categorical sense, with obvious projections

$\pi_1 : (x, y) \mapsto x$ and $\pi_2 : (x, y) \mapsto y$.

A.3 Cartesian Closed Categories

In **Set**, we have the set $\mathcal{P}X$ containing all subsets of X. In a general category $\mathbb{C}$, this is referred to as the *power object* of an object X. Power objects require the category to have a terminal object, finite products, a subobject classifier Ω, and finally, an exponent:

Definition A.12. Let Y be an object of a category $\mathbb{C}$ with products. A functor $(-)^Y$ is called an *exponent* functor if it is a right adjoint of the product functor $(-) \times Y$:

$$(-) \times Y \dashv (-)^Y$$

That is, for all objects X and Z of $\mathbb{C}$, we have an isomorphism

$$\hom_{\mathbb{C}}(X \times Y, Z) \xrightarrow{\alpha_{X,Z}} \hom_{\mathbb{C}}(X, Z^Y).$$

that is natural in X and Z. ■

The isomorphism α being natural in X and Z means that, if $X' \xrightarrow{f} X$ and $Z \xrightarrow{g} Z'$ are morphisms in $\mathbb{C}$, then

$$\begin{tikzcd}
\hom_{\mathbb{C}}(X \times Y, Z) \arrow[r, "\alpha_{X,Z}"] \arrow[d, "g \circ (-) \circ [f, 1_Y]"] & \hom_{\mathbb{C}}(X, Z^Y) \arrow[d, "g^Y \circ (-) \circ f"] \\
\hom_{\mathbb{C}}(X' \times Y, Z') \arrow[r, "\alpha_{X',Z'}"] & \hom_{\mathbb{C}}(X', Z'^Y)
\end{tikzcd}$$

commutes.

Definition A.13. A category $\mathbb{C}$ is called *Cartesian closed* if it has a terminal object and, for every two objects X, Y, a binary product $X \times Y$ and an exponent X^Y. ■

A.4 Monads

Finally, we define the notions of pointed endofunctors, copointed endofunctors and monads.

Definition A.14. A pointed endofunctor (F, η) consists of a functor $F : \mathbb{C} \longrightarrow \mathbb{C}$ along with a unit $\eta : \mathrm{Id}_{\mathbb{C}} \Rightarrow F$. ■

Dually, we have:

Definition A.15. A copointed endofunctor (F, ϵ) consists of a functor $F : \mathbb{C} \longrightarrow \mathbb{C}$ along with a counit $\epsilon : F \Rightarrow \mathrm{Id}_{\mathbb{C}}$. ■

Pointed endofunctors can be accompanied by a multiplication $\mu : FF \Rightarrow F$ to obtain a monad:

Definition A.16. A monad (F, μ, η) consists of a functor $F : \mathbb{C} \longrightarrow \mathbb{C}$ along with a multiplication $\mu : FF \Rightarrow F$ and a unit $\eta : \mathrm{Id}_{\mathbb{C}} \Rightarrow F$, such that the diagrams

$$\begin{tikzcd}
FFFX \arrow[r, "F\mu_X"] \arrow[d, "\mu_{FX}"] & FFX \arrow[d, "\mu_X"] \\
FFX \arrow[r, "\mu_X"] & FX
\end{tikzcd}
\qquad
\begin{tikzcd}
FX \arrow[r, "F\eta_X"] \arrow[dr, "\mathrm{id}_{FX}"'] & FFX \arrow[d, "\mu_X"] & FX \arrow[l, "\eta_{FX}"'] \arrow[dl, "\mathrm{id}_{FX}"] \\
 & FX &
\end{tikzcd}$$

commute in $\mathbb{C}$ for every object X of $\mathbb{C}$. ■

B Proofs

B.1 Proof of Lemma 2.22

Lemma. Let Σ be an algebraic signature, and $S : \mathbf{Set} \longrightarrow \mathbf{Set}$ a functor such that

$$SX = \coprod_{\mathsf{f}\in\Sigma} \underbrace{X \times \ldots \times X}_{\sharp\mathsf{f}}$$

The set $\Sigma^*\emptyset$ of closed Σ-terms carries an initial S-algebra structure.

Proof. We assume that Σ contains at least one constant, since otherwise, $S\Sigma^*\emptyset = \Sigma^*\emptyset = \emptyset$, and all functions are trivial.

Denote the coproduct injections into $\coprod_{\mathsf{f}\in\Sigma}(\Sigma^*\emptyset)^{\sharp\mathsf{f}}$ by ι_f.

We let $\alpha : \coprod_{\mathsf{f}\in\Sigma}(\Sigma^*\emptyset)^{\sharp\mathsf{f}} \longrightarrow \Sigma^*\emptyset$ be the algebra structure consisting of components $\alpha_\mathsf{f} : (\Sigma^*\emptyset)^{\sharp\mathsf{f}} \longrightarrow \Sigma^*\emptyset$ defined as follows (we use $1 = \{*\}$ here):

$$\alpha_\mathsf{f} : \begin{cases} \iota_\mathsf{f}(*) \mapsto \mathsf{f} & \text{if } \sharp\mathsf{f} = 0, \\ \iota_\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}}) \mapsto \mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}}) & \text{if } \sharp\mathsf{f} > 0. \end{cases}$$

Let (X,σ) be an S-algebra, and define a function $g : \Sigma^*\emptyset \longrightarrow X$ recursively by:

$$g : \begin{cases} \mathsf{f} \mapsto \sigma(\iota_{\mathsf{f},X}(*)) & \text{if } \sharp\mathsf{f} = 0, \\ \mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}}) \mapsto \sigma(\iota_{\mathsf{f},X}(g(e_1),\ldots,g(e_{\sharp\mathsf{f}}))) & \text{if } \sharp\mathsf{f} > 0. \end{cases}$$

where $\iota_{\mathsf{f},X}$ denotes the injection into $\coprod_{\mathsf{f}\in\Sigma} X^{\sharp\mathsf{f}}$ corresponding to $\mathsf{f} \in \Sigma$. Then the diagram

$$\begin{tikzcd} S\Sigma^*\emptyset \arrow[r, "Sg"] \arrow[d, "\alpha"] & SX \arrow[d, "\sigma"] \\ \Sigma^*\emptyset \arrow[r, "g"] & X \end{tikzcd}$$

commutes, since

$$(g \circ \alpha_\mathsf{f})(\iota_\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}})) = g(\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}})) = \sigma(\iota_{\mathsf{f},X}(g(e_1),\ldots,g(e_{\sharp\mathsf{f}}))) = (\sigma \circ Sg)(\iota_\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}})),$$

showing that g is a morphism of S-algebras.

To show uniqueness of g, we recursively define the following function $N : \Sigma^*\emptyset \longrightarrow \mathbb{N}$, giving the maximum operator depth of a term in $\Sigma^*\emptyset$:

$$N : \begin{cases} \mathsf{f} \mapsto 0 & \text{if } \sharp\mathsf{f} = 0, \\ \mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}}) \mapsto 1 + \max\Big\{N(e_i) : i \in \{1,\ldots,\sharp\mathsf{f}\}\Big\} & \text{if } \sharp\mathsf{f} > 0. \end{cases}$$

Let $g' : \Sigma^*\emptyset \longrightarrow X$ be another morphism of S-algebras. We prove that $g = g'$, i.e., that $g(\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}})) = g'(\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}}))$ for every term $\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}})$ in $\Sigma^*\emptyset$, by strong induction on the maximum operator depth $N(\mathsf{f}(e_1,\ldots,e_{\sharp\mathsf{f}}))$.

For the base case with $N(\mathsf{f}) = 0$, consisting of all constants f, indeed, we have

$$g'(\mathsf{f}) = (g' \circ \alpha_{\mathsf{f}})(\iota_{\mathsf{f}}(*)) = (\sigma \circ Sg')(\iota_{\mathsf{f}}(*)) = \sigma(\iota_{\mathsf{f},X}(*)) = g(\mathsf{f}).$$

Here, we use that the component of Sg' for f is the unique function $1 \longrightarrow 1$.

For the induction step, consider a term $\mathsf{f}(e_1,\dots e_{\sharp\mathsf{f}})$ with operator depth $n := N(\mathsf{f}(e_1,\dots e_{\sharp\mathsf{f}}))$. The induction hypothesis is that, for all terms of operator depths k such that $k < n$, the images under g and g' are equal. Then, in particular, by definition of N, $g(e_i) = g'(e_i)$ for all $i \in \{1,\dots,\sharp\mathsf{f}\}$.

$$\begin{aligned}
g'(\mathsf{f}(e_1,\dots e_{\sharp\mathsf{f}})) &= (g' \circ \alpha_{\mathsf{f}})(\iota_{\mathsf{f}}(e_1,\dots,e_{\sharp\mathsf{f}})) && \text{by definition of } \alpha_{\mathsf{f}} \\
&= (\sigma \circ Sg')(\iota_{\mathsf{f}}(e_1,\dots,e_{\sharp\mathsf{f}})) && \text{algebra morphism } g' \\
&= \sigma(\iota_{\mathsf{f},X}(g'(e_1),\dots,g'(e_{\sharp\mathsf{f}}))) && \text{definition of } Sg' \\
&= \sigma(\iota_{\mathsf{f},X}(g(e_1),\dots,g(e_{\sharp\mathsf{f}}))) && \text{I.H.} \\
&= (\sigma \circ Sg)(\iota_{\mathsf{f}}(e_1,\dots,e_{\sharp\mathsf{f}})) && \text{definition of } Sg \\
&= (g \circ \alpha_{\mathsf{f}})(\iota_{\mathsf{f}}(e_1,\dots,e_{\sharp\mathsf{f}})) && \text{algebra morphism } g \\
&= g(\mathsf{f}(e_1,\dots e_{\sharp\mathsf{f}})) && \text{by definition of } \alpha_{\mathsf{f}}
\end{aligned}$$

We conclude that $g = g'$ and hence, that α is initial. ☺

B.2 Proof of Lemma 3.6

Lemma. For any monoid M, the category M-**Set** is cartesian closed.

This proof is dedicated to the memory of Harold Simmons, whose unpublished notes [22] were absolutely indispensable.

Proof. As we already saw, the product is as in set, where the action of M is just pointwise action on both elements of a pair.

To show that M-**Set** is cartesian closed, we should construct an *exponent*, i.e., for a fixed M-set Y, we should give a functor $(-)^Y$ that is a right adjoint to the product $(-) \times Y$:

$$(-) \times Y \dashv (-)^Y.$$

For this, define, as a set,

$$Z^Y := \hom_{M\text{-}\mathbf{Set}}(M \times Y, Z).$$

In other words, Z^Y contains all equivariant functions from $M \times Y$ to Z. Here, M is an M-set with action $m \cdot n = mn$. We equip this set Z^Y with an action of M by letting, for $f \in Z^Y$ and $m \in M$, $m \cdot f = f^m$, where

$$f^m : M \times Y \longrightarrow Z, \quad (n, y) \mapsto (nm, y)$$

We should check that f^m is equivariant: indeed, for $n, n' \in M$ and $y \in Y$, we have

$$n' \cdot f^m(n, y) = n' \cdot f(nm, y) = f(n'nm, n'y) = f^m(n'n, n'y)$$

where we use that f is equivariant. This indeed defines an action of M on Z^Y, since $e \cdot f = f$ for all $f \in Z^Y$ and $m \cdot (m' \cdot f) = (mm') \cdot f$, since, for all $(n, y) \in M \times Y$, we have

$$(m \cdot (m' \cdot f))(n, y) = (m' \cdot f)(nm, y) = f(nmm', y) = (mm' \cdot f)(n, y)$$

We conclude that Z^Y is an M-set with the action given by $m \cdot f = f^m$.

The next step is to show that exponentiation defined this way is indeed a right adjoint to the product. For this, we establish an isomorphism $\hom(X \times Y, Z) \cong \hom(X, Z^Y)$ that is natural in X and Z. We let

$$\begin{tikzcd} \hom_{M\text{-}\mathbf{Set}}(X \times Y, Z) \arrow[r, shift left, "(\cdot)^\sharp"] & \hom_{M\text{-}\mathbf{Set}}(X, Z^Y) \arrow[l, shift left, "(\cdot)_\flat"] \end{tikzcd}$$

by defining, for all $f \in \hom(X \times Y, Z)$ and all $g \in \hom(X, Z^Y)$,

$$\begin{array}{llllll} f^\sharp : & X & \longrightarrow & Z^Y, & x & \mapsto \big(M \times Y \longrightarrow Z, \quad (m, y) \mapsto f(m \cdot x, y)\big) \\ g_\flat : & X \times Y & \longrightarrow & Z, & (x, y) & \mapsto g(x)(e, y) \end{array}$$

Put differently: for all $x \in X$, $y \in Y$, and $m \in M$:

$$f^\sharp(x)(m, y) = f(m \cdot x, y) \quad \text{and} \quad g_\flat(x, y) = g(x)(e, y)$$

We want $(\cdot)^\sharp$ and $(\cdot)_\flat$ to be well-defined, so we should check that $f^\sharp$ and $g_\flat$ are equivariant functions for all f, g:

- If $f^\sharp$ is to be an equivariant function from X to Z^Y, we should first check that $f^\sharp(x)$ indeed defines an element of Z^Y, i.e., we should show that $f^\sharp(x)$ is an equivariant function from $M \times Y$ to Z for all $x \in X$. Indeed, for all $x \in X$, $m, n \in M$ and $y \in Y$:

$$\begin{aligned} n \cdot (f^\sharp(x)(m,y)) &= n \cdot f(m \cdot x, y) && \{\text{definition of } f^\sharp\} \\ &= f(n \cdot (m \cdot x), n \cdot y) && \{\text{equivariance of } f\} \\ &= f(nm \cdot x, n \cdot y) \\ &= f^\sharp(x)(nm, n \cdot y) && \{\text{definition of } f^\sharp\} \end{aligned}$$

So $f^\sharp(x)$ is equivariant for all $x \in X$. Finally, note that, for all $(m,y) \in M \times Y$,

$$\begin{aligned} (n \cdot f^\sharp(x))(m,y) &= f^\sharp(x)(mn, y) && \{\text{action of } n \text{ on } f^\sharp\} \\ &= f(mn \cdot x, y) && \{\text{definition of } f^\sharp\} \\ &= f(m \cdot (n \cdot x), y) \\ &= f^\sharp(n \cdot x)(m, y) \end{aligned}$$

Hence, as functions, $n \cdot f^\sharp(x)$ and $f^\sharp(n \cdot x)$ are equal. We conclude that $f^\sharp$ is equivariant.

- Also $g_\flat$ is equivariant:

$$\begin{aligned} m \cdot g_\flat(x,y) &= m \cdot (g(x)(e,y)) && \{\text{definition of } g_\flat\} \\ &= g(x)(m \cdot e, m \cdot y) && \{g(x) \in Z^Y \text{ is equivariant}\} \\ &= (m \cdot g(x))(e, m \cdot y) && \{\text{action of } m \text{ on } g(x) \text{ in } Z^Y\} \\ &= g(m \cdot x)(e, m \cdot y) && \{g \text{ is equviariant}\} \\ &= g_\flat(m \cdot x, m \cdot y) && \{\text{definition of } g_\flat\} \end{aligned}$$

We conclude that the mappings $(\cdot)^\sharp$ and $(\cdot)_\flat$ are well-defined.

Observe that $(f^\sharp)_\flat = f$ and $(g_\flat)^\sharp = g$: for all $x \in X$, $y \in Y$, and $m \in M$, we have

$$(f^\sharp)_\flat(x,y) = f^\sharp(x)(e,y) = f(x,y)$$

and

$$(g_\flat)^\sharp(x)(m,y) = g_\flat(m \cdot x, y) = g(m \cdot x)(e,y) = (m \cdot g(x))(e,y) = g(x)(m,y)$$

Hence, the mappings $(\cdot)^\sharp$ and $(\cdot)_\flat$ are each other's inverses and they establish the bijection $\hom(X \times Y, Z) \cong \hom(X, Z^Y)$.

As a final step in showing that exponentiation as we defined is a right adjoint to the product, we show that this bijection is natural in X and Z. For this, let $X' \xrightarrow{a} X$ and $Z \xrightarrow{c} Z'$ be morphisms, and we show that the following two diagrams commutes:

$$\begin{tikzcd} \hom_{M\text{-}\mathbf{Set}}(X \times Y, Z) \arrow[r, shift left, "(\cdot)^\sharp"] \arrow[d, "c \circ (-) \circ [a, 1_Y]"] & \hom_{M\text{-}\mathbf{Set}}(X, Z^Y) \arrow[l, shift left, "(\cdot)_\flat"] \arrow[d, "c^Y \circ (-) \circ a"] \\ \hom_{M\text{-}\mathbf{Set}}(X' \times Y, Z') \arrow[r, shift left, "(\cdot)^\sharp"] & \hom_{M\text{-}\mathbf{Set}}(X', Z'^Y) \arrow[l, shift left, "(\cdot)_\flat"] \end{tikzcd} \tag{12}$$

where $c^Y : Z^Y \longrightarrow Z'^Y, f \mapsto c \circ f$. First, observe that, for all $x' \in X'$ and $(m, y) \in M \times Y$,

$$\begin{aligned}
(c \circ f \circ [a, 1_Y])^\sharp(x')(m,y) &= (c \circ f \circ [a, 1_Y])(m \cdot x', y) && \{\text{definition of } f^\sharp\}\\
&= c(f(a(m \cdot x'), y)) \\
&= c(f(m \cdot a(x'), y)) && \{\text{equivariance of } a\}\\
&= c(f^\sharp(a(x'))(m,y)) && \{\text{definition of } f^\sharp\}\\
&= c^Y(f^\sharp(a(x')))(m,y) && \{\text{definition of } c^Y\}\\
&= (c^Y \circ f^\sharp \circ a)(x')(m,y)
\end{aligned}$$

This shows that, going from left to right, the diagram (12) commutes.

Going the other way, observe that, for all $(x', y) \in X' \times Y$,

$$\begin{aligned}
(c^Y \circ g \circ a)_\flat(x', y) &= [c^Y \circ g \circ a](x')(e, y) && \{\text{definition of } g_\flat\}\\
&= \big(c^Y(g(a(x')))\big)(e,y) \\
&= c\big(g(a(x'))(e,y)\big) && \{\text{definition of } c^Y\}\\
&= c\big(g_\flat(a(x'), y)\big) && \{\text{definition of } g_\flat\}\\
&= (c \circ g_\flat \circ [a, 1_Y])(x', y)
\end{aligned}$$

We conclude that (12) commutes both ways, and hence, for any object Y in M-**Set**:

$$(-) \times Y \dashv (-)^Y$$

Evaluation is now given by $ev := (1_{Z^Y})_\flat$, so $ev : Z^Y \times Y \longrightarrow Z, (f, y) \mapsto f(e, y)$. Indeed, the triangle

$$\begin{tikzcd}
X \times Y \arrow[d, "{[f^\sharp, 1_Y]}"'] \arrow[dr, "f"] & \\
Z^Y \times Y \arrow[r, "ev"'] & Z
\end{tikzcd}$$

commutes for any $X \times Y \xrightarrow{f} Z$, since $(ev \circ [f^\sharp, 1_Y])(x,y) = ev(f^\sharp(x), y) = f^\sharp(x)(e, y) = f(x,y)$. ☺

B.3 Proof of Proposition 3.8

Proposition. Let X and Y be G-sets. Then we have an isomorphism $Y^X \cong \hom_{\mathbf{Set}}(X,Y)$ of G-sets.

Proof. We define two equivariant functions $\begin{tikzcd} Y^X \arrow[r, shift left, "(\cdot)^{\downarrow}"] & \hom_{\mathbf{Set}}(X,Y) \arrow[l, shift left, "(\cdot)^{\uparrow}"] \end{tikzcd}$ and show that they are inverses.

Let $k \in Y^X$, so $k : G \times X \longrightarrow Y$ is equivariant. Define a function $k^{\downarrow} : X \longrightarrow Y$ by $k^{\downarrow}(x) = k(e,x)$. To see that $(\cdot)^{\downarrow}$ is equivariant, note that, for all $g \in G$ and $x \in X$,

$$\begin{aligned}
(g\cdot k)^{\downarrow}(x) &= (g\cdot k)(e,x) && \{\text{definition of } (\cdot)^{\downarrow}\}\\
&= k(e\cdot g,x) && \{\text{action of } Y^X\}\\
&= k(g\cdot e,x) && \{\text{natural action of } G \text{ (twice)}\}\\
&= k(g\cdot e, g\cdot(g^{-1}\cdot x) && \{\text{action of } X\}\\
&= g\cdot k(e,g^{-1}\cdot x) && \{\text{equivariance of } k\}\\
&= g\cdot k^{\downarrow}(g^{-1}\cdot x) && \{\text{definition of } (\cdot)^{\downarrow}\}\\
&= (g\cdot k^{\downarrow})(x) && \{\text{action of } \hom_{\mathbf{Set}}(X,Y)\}
\end{aligned}$$

Conversely, let $l \in \hom_{\mathbf{Set}}(X,Y)$ be a function of sets and define $l^{\uparrow} : G\times X \longrightarrow Y$ by $l^{\uparrow}(g,x) = (g\cdot l)(x)$. To see that indeed $l^{\uparrow} \in Y^X$, notice that $l^{\uparrow}$ is equivariant: for all $g,h \in G$ and $x \in X$,

$$\begin{aligned}
h\cdot l^{\uparrow}(g,x) &= h\cdot(g\cdot l)(x) && \{\text{definition of } (\cdot)^{\uparrow}\}\\
&= h\cdot(g\cdot l(g^{-1}\cdot x)) && \{\text{action of } \hom_{\mathbf{Set}}(X,Y)\}\\
&= hg\cdot l(g^{-1}h^{-1}h\cdot x)\\
&= hg\cdot l((hg)^{-1}\cdot h\cdot x)\\
&= (hg\cdot l)(h\cdot x) && \{\text{action of } \hom_{\mathbf{Set}}(X,Y)\}\\
&= l^{\uparrow}(hg,h\cdot x) && \{\text{definition of } (\cdot)^{\uparrow}\}\\
&= l^{\uparrow}(h\cdot g,h\cdot x) && \{\text{natural action of } G\}
\end{aligned}$$

Furthermore, $(\cdot)^{\uparrow}$ is equivariant, since, for all $g,h \in G$ and $x \in X$, we have

$$\begin{aligned}
(g\cdot l)^{\uparrow}(h,x) &= (h\cdot(g\cdot l))(x) && \{\text{definition of } (\cdot)^{\uparrow}\}\\
&= (hg\cdot l)(x)\\
&= l^{\uparrow}(hg,x) && \{\text{definition of } (\cdot)^{\uparrow}\}\\
&= l^{\uparrow}(h\cdot g,x) && \{\text{natural action of } G\}\\
&= (g\cdot l^{\uparrow})(h,x) && \{\text{action of } Y^X\}
\end{aligned}$$

We have established that $\begin{tikzcd} Y^X \arrow[r, shift left, "(\cdot)^{\downarrow}"] & \hom_{\mathbf{Set}}(X,Y) \arrow[l, shift left, "(\cdot)^{\uparrow}"] \end{tikzcd}$ are equivariant. It remains to show that they are inverses.

- For $k \in Y^X$, let $(g,x) \in G\times X$ be arbitrary. Then

$$(k^{\downarrow})^{\uparrow}(g,x) = (g\cdot k^{\downarrow})(x) = g\cdot k^{\downarrow}(g^{-1}\cdot x) = g\cdot k(e,g^{-1}\cdot x) = k(g\cdot e, g\cdot g^{-1}\cdot x) = k(g,x)$$

- Conversely, for $l \in \hom_{\mathbf{Set}}(X, Y)$, observe that, for $x \in X$, we have

$$(l^{\uparrow})^{\downarrow}(x) = l^{\uparrow}(e, x) = (e \cdot l)(x) = e \cdot l(e^{-1} \cdot x) = l(x)$$

Finally, we conclude that $Y^X \cong \hom_{\mathbf{Set}}(X, Y)$ as G-sets. ☺

B.4 Proof of Theorem 4.21

Theorem 4.20. There is a bijective correspondence between B-coalgebras that satisfy Rule ($\star$) and $\widetilde{B}$-coalgebras.

Proof. Recall that a B-coalgebra g is a map $X \longrightarrow \mathcal{P}_{\mathrm{fs}}LX$ with $L = \coprod_{i=1}^{t} \mathbb{A}^{k_i} \times X$, and a $\widetilde{B}$-coalgebra $\widetilde{g}$ is a map $X \longrightarrow \mathcal{P}_{\mathrm{fs}}\widetilde{L}X$ with $\widetilde{L} = \coprod_{i=1}^{t} \mathbb{A}^{k_i - \mathfrak{b}(i)} \times \left[\mathbb{A}^{\mathfrak{b}(i)}\right] X$.

To relieve us of some notation, we will assume $t = 1$. Conceptually, the proof can be easily extended to multiple coproduct terms.

It is also straightforward to generalize from a single binding name to multiple binding names: just assert freshness conditions over tuples instead of single names. Therefore, for the purpose of readability, we further assume that there is exactly one binding name in the single coproduct term. (For coproduct terms where there are no binding names, the correspondence is trivial.)

With these modifications, writing $W = \mathbb{A}^{k_1 - 1}$, we have

$$\widetilde{B^*}Y = \mathcal{P}_{\mathrm{fs}}\widetilde{L^*}Y \qquad \text{with} \qquad \widetilde{L^*}Y = W \times [\mathbb{A}]\, Y$$

and

$$B^*Y = \mathcal{P}_{\mathrm{fs}}L^*Y \qquad \text{with} \qquad L^*Y = W \times \mathbb{A} \times Y,$$

and Rule ($\star$) becomes

Rule ($\star^*$). Let (g, X) be a B^*-coalgebra. We say that g satisfies Rule ($\star^*$) if

$$\forall (w, b, y, x) \in W \times \mathbb{A} \times X \times X \,.\, (w, b, y) \in g(x) \implies b \# (w, x)$$

It will now suffice to demonstrate a bijective correspondence between B^*-coalgebras that satisfy Rule ($\star^*$) and $\widetilde{B^*}$-coalgebras.

The bijection we use is:

$$\begin{tikzcd} B^*\textbf{-Coalg} \arrow[r, shift left, "\phi"] & \widetilde{B^*}\textbf{-Coalg} \arrow[l, shift left, "\psi"] \end{tikzcd}$$

$$\begin{array}{rcl} (X, g) & \mapsto & (\widetilde{g} : x \mapsto \{(w, \langle b \rangle y) \mid (w, b, y) \in g(x)\}) \\ (g : x \mapsto \{(w, b', y') \approx_\alpha (w, b, y) \mid & \leftarrow\!\shortmid & (X, \widetilde{g}) \\ (w, \langle b \rangle y) \in \widetilde{g}(x), b' \# (w, x)\}) & & \end{array}$$

So we have $\phi : B^*\textbf{-Coalg} \longrightarrow \widetilde{B^*}\textbf{-Coalg}, (X, g) \mapsto (X, \widetilde{g})$, where

$$\widetilde{g} : x \mapsto \{(w, \langle b \rangle y) \in \widetilde{L}X \mid (w, b, y) \in g(x)\}$$

Intuitively, an image of x under $\widetilde{g}$ is less dense than the image under g. This is because we can have $(w, \langle b \rangle y) = (w, \langle b' \rangle y')$ for many $(w, b, y), (w, b', y') \in g(x)$.

Note that $\widetilde{g}(x)$ is finitely supported for every x, since $g(x)$ is, and we claim furthermore that $\widetilde{g}$ is equivariant (establishing ϕ as a well-defined mapping). Indeed, for $\pi \in \mathrm{Pm}(\mathbb{A})$:

- If $\widetilde{z} \in \widetilde{g}(\pi \cdot x)$, then $\widetilde{z} = (w, \langle b \rangle y)$ for some $(w, b, y) \in g(\pi \cdot x) = \pi \cdot g(x)$. Therefore, we have some (w', b', y') with $(w, b, y) = \pi \cdot (w', b', y')$, and $(w', b', y') \in g(x)$. Then $(w', \langle b' \rangle y') \in \widetilde{g}(x)$, and therefore

  $$\pi \cdot (w', \langle b' \rangle y') = (\pi \cdot w', \langle \pi b' \rangle (\pi \cdot y')) = (w, \langle b \rangle y) = \widetilde{z} \in \pi \cdot \widetilde{g}(x)$$

- Conversely, if $\widetilde{z} \in \pi \cdot \widetilde{g}(x)$, then $\widetilde{z} = \pi \cdot (w, \langle b \rangle y)$ for some $(w, \langle b \rangle y) \in \widetilde{g}(x)$, so $\widetilde{z} = (\pi \cdot w, \langle \pi b \rangle (\pi \cdot y))$. By definition of $\widetilde{g}$, this means that we have some $z = (w, b', y') \in g(x)$ with $(b', y') \approx_\alpha (b, y)$. Since g is equivariant, we have $\pi \cdot z = (\pi \cdot w, \pi b', \pi \cdot y') \in \pi \cdot g(x) = g(\pi \cdot x)$. By definition of $\widetilde{g}$, then,

  $$(\pi \cdot w, \langle \pi b' \rangle (\pi \cdot y')) \in \widetilde{g}(\pi \cdot x)$$

  Since $(b', y') \approx_\alpha (b, y)$ and by equivariance of $\approx_\alpha$, we have $(\pi b', (\pi \cdot y')) \approx_\alpha (\pi b, (\pi \cdot y))$. But then $(\pi \cdot w, \langle \pi b' \rangle (\pi \cdot y')) = (\pi \cdot w, \langle \pi b \rangle (\pi \cdot y)) = \widetilde{z}$, and therefore, $\widetilde{z} \in \widetilde{g}(\pi \cdot x)$.

We conclude that $\widetilde{g}$ is equivariant, and therefore a $\widetilde{B}^*$-coalgebra structure.

The converse mapping of coalgebras requires a slightly more sophisticated construction. Indeed, for a coalgebra $(X, \widetilde{g})$, we need, for every $(w, \langle b \rangle y) \in \widetilde{g}(x)$, a set of triples (w, b', y'), where (b', y') ranges over the equivalence class (which is a set) of (b, y).

So define $\psi : \widetilde{B}^*\text{-}\mathbf{Coalg} \longrightarrow B^*\text{-}\mathbf{Coalg}, (X, \widetilde{g}) \mapsto (X, g)$, where

$$g : x \mapsto \{(w, b', y') \in LX \mid (b', y') \approx_\alpha (b, y), \text{ where } (w, \langle b \rangle y) \in \widetilde{g}(x) \text{ and } b' \# \{w, x\}\}$$

That is, the image of x under g contains all equivalence classes of the $\langle b \rangle y$ in $\widetilde{g}(x)$ (paired with w), but restricted to those triples where the bound name is fresh in $\{w, x\}$.

Once again, we need equivariance of g, so let $\pi \in \mathrm{Pm}(\mathbb{A})$.

- If $z \in g(\pi \cdot x)$, then $z = (w, b', y')$ where, for some $(w, \langle b \rangle y) \in \widetilde{g}(\pi \cdot x)$, we have $(b', y') \approx_\alpha (b, y)$ and $b' \# \{w, \pi \cdot x\}$. Observe that $\langle b \rangle y = \langle b' \rangle y'$ so $(w, \langle b' \rangle y') \in \pi \cdot \widetilde{g}(x)$ by equivariance of $\widetilde{g}$. Let $(w', b'', y'') = \pi^{-1} \cdot z = \pi^{-1} \cdot (w, b', y')$ and note that

  $$(w', \langle b'' \rangle y'') \in \pi^{-1} \cdot \pi \cdot \widetilde{g}(x) = \widetilde{g}(x) \tag{13}$$

  Furthermore, since # is equivariant, from applying π^{-1} to $b' \# \{w, \pi \cdot x\}$ we obtain $b'' \# \{w', x\}$. By definition of g, then, $(w', b'', y'') \in g(x)$ (use (13) and the fact that $\approx_\alpha$ is reflexive). Hence, $\pi \cdot (w', b'', y'') = z \in \pi \cdot g(x)$.

- Conversely, let $z \in \pi \cdot g(x)$. We can write $z = \pi \cdot (w, b', y')$ and there is $(w, \langle b \rangle y) \in \widetilde{g}(x)$ such that $(b, y) \approx_\alpha (b', y')$ and $b' \# \{w, x\}$. Note that $\langle b \rangle y = \langle b' \rangle y'$. By equivariance of #, we have $\pi b' \# \{\pi \cdot w, \pi \cdot x\}$. Furthermore, using equivariance of $\widetilde{g}$, we have that $(\pi \cdot w, \langle \pi b \rangle (\pi \cdot y)) = (\pi \cdot w, \langle \pi b' \rangle (\pi \cdot y')) \in \widetilde{g}(\pi \cdot x)$. By definition of g, then, $(\pi \cdot w, \pi b', \pi \cdot y') = z \in g(\pi \cdot x)$, where we use reflexivity of $\approx_\alpha$ again.

We conclude that g is a morphism and therefore a B^*-coalgebra. Also observe that g satisfies Rule ($\star^*$) by construction. It remains to show that the two constructed mappings ϕ and ψ are inverses of each other.

- Let (X, g) be a B^*-coalgebra satisfying Rule ($\star^*$), and write $(X, \widetilde{g}) = \phi((X, g))$ and $(X, g') = \psi(\phi((X, g)))$. Note that now also g' satisfies Rule ($\star^*$). We claim that $(X, g') = (X, g)$. To see this, observe that, for all $x \in X$:

    - If $(w,b,y) \in g(x)$, then $(w,\langle b\rangle y) \in \widetilde{g}(x)$. Since $(b,y) \approx_\alpha (b,y)$ and $b \# \{w,x\}$, by definition of ψ, we also have $(w,b,y) \in g'(x)$.
    - Conversely, if $(w,b',y') \in g'(x)$ then there is $(w,\langle b\rangle y) \in \widetilde{g}(x)$ such that $(b',y') \approx_\alpha (b,y)$ and $b' \# \{w,x\}$. But then $(w,\langle b\rangle y) = (w,\langle b'\rangle y') \in \widetilde{g}(x)$, and, by definition of ϕ, this in turn means that there is some $(w,b'',y'') \in g(x)$ such that $(b'',y'') \approx_\alpha (b',y')$. Apply Lemma 4.18 to conclude that then $(w,b',y') \in g(x)$.

  We conclude that $g(x) = g'(x)$ and since x was arbitrary, $(X,g) = (X,g')$.

- Now for the reverse composition of ϕ and ψ, let $(X,\widetilde{g})$ be a $\widetilde{B}^*$-coalgebra. Define $(X,g) = \psi((X,\widetilde{g}))$ and $(X,\hat{g}) = \phi(\psi((X,\widetilde{g})))$. The claim is now that $(X,\widetilde{g}) = (X,\hat{g})$. Let $x \in X$ be arbitrary.
    - Let $(w,\langle b\rangle y) \in \widetilde{g}(x)$. Choose some $b' \# \{w,x\}$ and let $y' = (b\ b') \cdot y$. Note that by Lemma 3.29, we have $(b,y) \approx_\alpha (b',y')$ (in the case that $b = b'$ as well as $b \neq b'$). Hence, by definition of ψ, we have $(w,b',y') \in g(x)$. Conclude that $(w,\langle b'\rangle y') = (w,\langle b\rangle y) \in \hat{g}(x)$.
    - Conversely, if $(w,\langle b\rangle y) \in \hat{g}(x)$, then there is $(w,b',y') \in g(x)$ such that $(b',y') \approx_\alpha (b,y)$ and $b' \# \{w,x\}$. By definition of ψ, this in turn means that there is $(w,\langle b''\rangle y'') \in \widetilde{g}(x)$ such that $(b'',y'') \approx_\alpha (b',y')$. But obviously $\langle b''\rangle y'' = \langle b\rangle y$, where we use transitivity of $\approx_\alpha$, and hence, $(w,\langle b\rangle y) \in \widetilde{g}(x)$.

  So we see that $\widetilde{g}(x) = \hat{g}(x)$, for all $x \in X$. Therefore, $(X,\widetilde{g}) = (X,\hat{g})$ as coalgebras.

Finally, we conclude that ϕ and ψ establish the bijective correspondence between B^*-coalgebras satisfying Rule ($\star^*$) and $\widetilde{B}^*$-coalgebras.

If we consider b to be tuples of names and let coproduct terms of L and $\widetilde{L}$ correspond naturally, the statement in Theorem 4.21 will follow. ☺

References

[1] L. Aceto, I. Fábregas, Á. García-Pérez, A. Ingólfsdóttir, and Y. Ortega-Mallén. Rule formats for nominal process calculi. In *28th International Conference on Concurrency Theory (CONCUR 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[2] L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In *Handbook of process algebra*, pages 197–292. Elsevier, 2001.

[3] F. Bartels. On generalised coinduction and probabilistic specification formats: Distributive laws in coalgebraic modelling, 2004.

[4] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM (JACM)*, 42(1):232–268, 1995.

[5] R. Clouston. Generalised name abstraction for nominal sets. In F. Pfenning, editor, *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7794 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2013.

[6] L. Eriksson. Weak nominal modal logic. In *Formal Techniques for Distributed Objects, Components, and Systems: 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, volume 10321, page 179. Springer, 2017.

[7] M. P. Fiore and S. Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 49–58. IEEE Computer Society, 2006.

[8] M. J. Gabbay and M. Hofmann. Nominal renaming sets. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 158–173. Springer, 2008.

[9] H. P. Gumm and T. Schröder. Monoid-labeled transition systems. *Electronic Notes in Theoretical Computer Science*, 44(1):185–204, 2001.

[10] B. Klin. Bialgebras for structural operational semantics: An introduction. *Theoretical Computer Science*, 412(38):5043–5069, 2011.

[11] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. In H. Reichel, editor, *Coalgebraic Methods in Computer Science, CMCS 2000, Berlin, Germany, March 25-26, 2000*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 230–260. Elsevier, 2000.

[12] S. MacLane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 2012.

[13] R. Milner et al. *A calculus of communicating systems*. 1980.

[14] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.

[15] J. Moerman and J. Rot. Separation and Renaming in Nominal Sets. In M. Fernández and A. Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[16] M. Mousavi, M. A. Reniers, and J. F. Groote. Sos formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3):238–272, 2007.

[17] U. Nestmann. Welcome to the jungle: A subjective guide to mobile process calculi. In *International Conference on Concurrency Theory*, pages 52–63. Springer, 2006.

[18] A. M. Pitts. *Nominal sets: Names and symmetry in computer science*, volume 57. Cambridge University Press, 2013.

[19] J. J. Rutten. Universal coalgebra: a theory of systems. *Theoretical computer science*, 249(1):3–80, 2000.

[20] D. Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(1-2):235–274, 1996.

[21] D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

[22] H. Simmons. The topos actions on a monoid, 2003. Unpublished, accessed: 18 Aug 2021.

[23] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 280–291. IEEE Computer Society, 1997.

[24] T. Weber, R. F. Gutkovas, L.-H. Eriksson, J. Borgström, and J. Parrow. Modal logics for nominal transition systems. *Logical Methods in Computer Science*, 17, 2021.

[25] T. Wißmann. Minimality notions via factorization systems. In *Proc. 9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, 09 2021.