Accessible functors and final coalgebras for named sets

Vincenzo Ciancia

Supervisor
Prof. Ugo Montanari
Abstract

In the field of programming language semantics and concurrency theory, wide attention is paid to the so called name-passing calculi, i.e. formalisms where name generation and passing play a fundamental role. A prototypical example is provided by the $\pi$-calculus.

The peculiarities of name passing required to refine existing theoretical models and to invent new ones, such as coalgebras over presheaf categories. The theory of name passing has proven difficult to be used in applications, since many problems arise due to the presence of fresh names. For example, only a few specialised tools exist for automated verification of nominal calculi, such as the mobility workbench or mihda, the latter exploiting a model of computation with local names, called history-dependent automata, defined as coalgebras in the category of named sets. History dependent automata have been successful in modelling a certain number of formalisms with name passing. However, there has always been a gap between the definitions on presheaf categories, exploiting mathematical tools such as accessible functors, and definitions of coalgebras on named sets, that are given for each language in an ad-hoc way, often tied to implementation purposes.

In this thesis work we try to fill this gap, by linking history-dependent automata with the theoretical results that ensure correctness and full abstractness of the semantics of calculi in presheaf categories. In particular, we define a meta-language of accessible endofunctors in the category of named sets, that can be used to define the semantics of calculi in a modular way. We show how locality of names is reflected in mathematical properties of the functors, in a way that is close to intuition and common practice related to local names themselves. We also provide a coalgebraic characterisation of the semantics of the $\pi$-calculus as a finitely branching system, making sense in the general case of a representation technique that was used in [FMT05a] to minimise finite-state $\pi$-calculus agents.
Contents

I Introductory Material 9

1 Introduction 11

2 Background 25

2.1 Nominal Sets as Algebras for the Permutation Signature . . . . . . . . . . . . . . 25
2.2 Other Presheaf Models and Equivalence Results . . . . . . . . . . . . . . . . . . . . 28
2.3 Theory Morphisms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
2.4 Multi-colimits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
2.5 Coalgebras, Bisimulation and Accessibility . . . . . . . . . . . . . . . . . . . . . 32
2.6 Final Coalgebras for Accessible Endofunctors . . . . . . . . . . . . . . . . . . 34
2.7 The $\pi$-calculus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
  2.7.1 Syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
  2.7.2 Early Operational Semantics and Bisimulation . . . . . . . . . . . . . . . 37
2.8 Permutation Algebras for the $\pi$-calculus . . . . . . . . . . . . . . . . . . . 38

II Permutation Algebras 41

3 Behavioural Functors for Permutation Algebras 45

3.1 Some Results on Permutation Algebras . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.2 Name Abstraction and Concretion . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
  3.2.1 The Name Abstraction Functor . . . . . . . . . . . . . . . . . . . . . . . 48
  3.2.2 Properties of Abstraction . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
3.3 The Left Adjoint to $\delta$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
  3.3.1 The Counit as a concretion operation . . . . . . . . . . . . . . . . . . . . 57
3.4 Comparison with other definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
0.0. CONTENTS

Bibliography 125
Part I

Introductory Material
Chapter 1

Introduction

Nominal calculi

The research on the formal semantics of concurrent and interactive systems reached a turning point, at the end of the eighties, with the definition of the $\pi$-calculus by Milner, Parrow and Walker [MPW92]. The $\pi$-calculus is a language designed for mobility, with constructs for parallel composition, nondeterministic choice and fresh name generation. All previous languages for concurrent or interactive systems, such as Petri nets [Rei85] or the CCS [Mil80], lacked the fundamental ability to dynamically reconfigure the communication structure. This is an essential feature when representing mobility, because it allows processes to switch their communication channels while communication is in progress, without having to interrupt (and establish again) their connection.

Models for concurrent mobile processes are undoubtedly a need in today’s computer science, due to the increasing number of systems and protocols running on mobile hardware, but also to the massive usage of code migration mechanisms in distributed and network programming.

The novel approach of the $\pi$-calculus was to include both the transmission of channels as messages, which gives dynamic reconfiguration, and the restriction operator already found in the CCS. The latter models the generation of a fresh, private channel that can then be communicated to other processes. The $\pi$ calculus is based on pure names, in the sense that a name does not designate additional data: it is just an abstract notion whose only property is identity.

As Milner himself states [Mil03], names are the most important feature of the $\pi$-
calculus. The conjunction of this ability with fresh name generation (or restriction) is responsible for much of the expressive power of this language. Using restriction, it is possible to atomically communicate entire sequences of messages, to encode function calls and thus the $\lambda$-calculus [Mil92], to model causal dependencies between events [BS95], and to represent the unique identity of objects in object-oriented programming as in [Wal95], where a complete encoding of a parallel object-oriented language into the $\pi$-calculus is introduced. Another key use for fresh name generation is expressing secrecy properties in secure communication protocols: the spi-calculus, developed by Abadi and Gordon in 1996 [AG97], is an extension of the $\pi$-calculus designed for the description and analysis of cryptographic protocols. Other possibilities include the handling of sessions in service oriented computing, modelling properties related to memory allocation in programming languages and so on.

Following the definition of $\pi$-calculus, a great number of variants, extensions and related languages have been invented, and there is an entire family of process calculi for mobile processes, each one giving different visibility in the operational semantics to particular characteristics such as locations or resource usage of systems.

Once the power of name generation was understood, there was the need to provide adequate models to give a semantical and logical representation of these languages. Ordinary set theory, and systems built over it (in particular, labelled transition systems) proved to be at a lower level of abstraction, lacking the notion of fresh name generation along transitions. This is reflected in the fact that the LTS semantics of the $\pi$-calculus has side conditions related to the freshness of names, that give rise to a non-standard notion of bisimulation. Semantic equivalence, thus, does not coincide with (the standard notion of) equivalence of models: the semantics is not fully abstract. As a very practical implication, the lack of full abstractness makes difficult to adopt known techniques to augment the defined semantics with e.g. formal methods, known algorithms that manipulate the semantics, or logical characterisations (think of methodologies that work on the final semantics, such as coalgebraic minimisation, verification by bisimulation, or Stone duality for transition systems [BK05]).

Traditional logical methods, and in particular modal logics, also proved inadequate to reason about name passing, leading to the definition of logics with name binding, of which the most representatives are defined in [MPW93] and [Dam93].

Even on the side of the syntax, the problem of representing binding in a satisfying
way has been long standing, the most famous attempt being that of De Bruijn indexes [dB72]. The problem was tackled in the works on nominal syntax using functor categories [FPT99] or nominal sets [GP99, GP02], the latter aimed at providing a logical framework in which to represent axioms on terms up to α-equivalence [Pit01].

For the operational semantics, new models where developed, noticeably using coalgebras over functor categories, such as [FMS96, CSW97, FT01], or over permutation algebras [MP00, MP05]. Finality of the semantics in many cases ensures full abstractness, thus making these models appealing as a solid ground to develop theoretical results about nominal formalisms.

Elegant formalisation and nice full abstractness results are the strong point of all these models, that however lack the ability to discard no longer used fresh names. Algorithms that manipulate the semantics (e.g. bisimulation checking and model checking) typically work by reducing the system to finite states. In absence of garbage collection, finite state systems are obtained only for very simple processes. As an explanatory example, consider a program that generates a fresh channel \(c\), sends it to another process on channel \(d\), and then restarts its cycle using channel \(c\) in place of \(d\). This can be seen as an idealised version of a key-changing protocol. Despite its trivial formulation, this system has one state for each available fresh channel, that is, it has infinite states.

The problem of correctly modelling garbage collection has traditionally been an hurdle in the implementation of tools supporting formal methods for nominal calculi. For example, most state-of-the-art model checkers do not deal with resource generation at all, and the user is required to simulate this feature by defining a domain of names which is “big enough” to provide possible counterexamples. Basically, this limits fresh resources to a finite number that can not be known \textit{a priori}, introducing a source of incorrectness in verification.

It is frequent to tackle this problem by giving a “local” meaning to names: instead of giving to each name a defined meaning, names are just considered placeholders for resources, whose identity is established by the history of computation, by associating to each computation step a relabelling that binds names of the destination state to names of the source. To illustrate the concept, an analogy can be used with memory management in programming languages that feature garbage collection: the at each step of the computation, an injective relocation function may change the meaning of all the memory addresses.
This naive approach works in many cases. However, such models can be subtly fallacious. For example, the presence of redundant names may give rise to incorrect translation of models. Complications arise when intrinsic symmetries of the semantics are ignored (e.g., by representing just lists or sets of names). This can result in too concrete models if one is able to distinguish resources whose difference is not observable in the semantics, due to symmetry, or it can create an unnecessary combinatorial explosion in the number of analysed states in finite state methods.

**History-dependent automata**

The category of named sets and its coalgebras, history-dependent automata, were invented as a formal methodology to reason about nominal calculi, and to implement finite-state algorithms dealing with resource allocation.

History-dependent automata (HD-automata for short) started as an operational model, strongly aimed to verification purposes, and influenced by the history-preserving semantics of Petri nets [BDKP91]. The representation technique of HD-automata was first introduced in [MPY96] for the location semantics of CCS. In [MP97], the same idea has been applied to Petri nets. In these two simpler cases, the semantics given using HD-automata could be unfolded into finite labelled transition systems, allowing ordinary verification and minimisation algorithms to be re-used. In these works, category theory is still not used as a formal tool. Presented results are fully abstract with respect to the location semantics of CCS and the causal semantics of place-transition Petri nets, respectively.

Based on the success of these first attempts, a step towards verification of the $\pi$-calculus was done with the $\pi$-logic and the JACK verification environment [GR97]. Also there, an unfolding towards ordinary automata is used in the effective algorithm. These earlier versions of HD-automata for the $\pi$-calculus featured sets of names attached to states, and injective relabelings along the transition function. This resulted in a too concrete model, lacking full abstractness of the semantics, and existence of a unique minimal model.

In Marco Pistore’s thesis [Pis99], symmetries were added to the model. This choice was driven by the idea of permutation algebras and the coalgebraic, fully abstract semantics of the $\pi$-calculus of [MP00]. The main achievement obtained by introducing symmetries is existence of a unique minimal model for finite state $\pi$-calculus agents.
Still, HD-automata were viewed as just an operational model with a very ad-hoc definition. A first step towards a coalgebraic formulation had indeed been done in [MP98], by viewing HD-automata bisimulation as induced by open maps. However that characterisation was only for the case without symmetries. On the other hand, a coalgebraic presentation of the HD-automaton (with symmetries) for the $\pi$-calculus has been given in [FMP02] and used in [FMT05a] to drive the implementation of a minimisation procedure for finite state $\pi$-calculus agent, the mihda toolkit. The definition of bisimulation used therein is not the coalgebraic notion, due to the necessity of establishing a map of local names between two systems, which is typical of HD-automata. The minimisation procedure is based on iteration along the terminal sequence, a coalgebraic generalisation of the Kanellakis-Smolka partition refinement algorithm [KS90].

Stemming from the application-oriented view of previous works, the definition of the functor in [FMP02] and [FMT05a] is specific for the $\pi$-calculus. The definition is obtained "by ingenious reasoning", guided by the understanding of the semantics of the $\pi$-calculus provided by [MP00]. Thus, the definition of the functor therein is rather technical. As a major consequence, it was difficult to understand if the employed functor had a final coalgebra. In [FMT05a], this was of little concern: to the aim of minimising the $\pi$-calculus, correctness of the definitions in the finite state case is sufficient.

Nevertheless, the result in [FMT05a] is potentially of great impact: as symmetries are part of the definition of HD-automata, the minimal system obtained by the partition refinement algorithm presented there has the greatest possible symmetry up-to bisimulation. The importance of modelling symmetries in concurrency and of symmetry reduction in verification is becoming an established fact (see e.g. [SGE00] and related works), and the idea of minimisation up-to bisimulation is in accordance with the idea of modal logics characterising bisimulation (e.g. [MPW93]). Named sets can then be used as a general algorithmic framework to deal with systems featuring symmetries.

A comparison of the various categorical models of name passing was done in [FS06] and [GMM06], resulting in the proof of equivalence of the categories of permutation algebras (hence nominal sets), the Schanuel topos (the full subcategory of pullback-preserving endofunctors in the presheaf category $\text{Set}^I$) and named sets. From a theoretical point of view, this result implies that there is no difference in adopting one of a number of equivalent formalisms. However, named sets enjoy nice algorithmic properties which are absent in all the other models. As we will explain in detail when dealing with name ab-
straction in named sets, these algorithmic benefits are due to the handling of names as
local, bindable resources, rather than constants which have a global, immutable meaning.

The equivalence results enable one to reason in parallel: on one side, proving properties
in presheaf categories, on the other side, implementing algorithms using named sets.
Reasoning simultaneously on two equivalent categories proves fruitful. It is generally
easier, for example, to give a semantics using categories with global names, rather than
to employ named sets directly. This is especially true for permutation algebras, whose
coalgebras are little more than LTSs enriched with name allocation primitives.

However, as we already mentioned, there is still a problem in doing so: functors for
HD-automata were defined in an ad-hoc way for each one of the many formalisms that
were modeled in [Pis99].

**Contribution of the thesis**

The main focus of this thesis is to extend the equivalence results of [FS06] and [GMM06]
to categories of coalgebras, by defining a language of accessible endofunctors on named
sets, and a final coalgebra theorem that allows fully abstract models for nominal calculi
to be coinductively defined. As an example, the semantics of the $\pi$-calculus is included
in our framework, thus we make sense of the definitions in [FMT05a].

The first part of the thesis is devoted to a systematic exposition of permutation
algebras and their functors, in particular, name abstraction. A name abstraction functor
was never defined explicitly for permutation algebras, even though it should be the
most relevant feature of a category that handles names and name allocation. We do
this exploiting a theory morphism, thus taking advantage of the algebraic definition
of the category. This constructively obtains a left adjoint to name abstraction, which
turns out to correspond (as one might expect) to the freshness relation of Gabbay and
Pitts [GP02]. Group-theoretical properties (namely, orbits and symmetries) of both
functors are explored. This is important in the light of the definition over the category
of named sets, which (after the aforementioned equivalence results) is built upon the
group theoretical properties of objects in $\text{FSA}_{\pi}$. 

The name abstraction functor is patterned after an idea that is common to all pre-
vious works on history-dependent automata: a De Bruijn alike [dB72] treatment of
binding by shifting names. In this thesis, we provide an answer to the question wether
this machinery is comparable to name abstraction in presheaf categories or binding in
nominal sets: in §4.1 we exhibit an abstract syntax for λ-calculus à la De Bruijn, as the initial algebra of a functor employing the name abstraction operator on permutation algebras. Other functors are presented and studied to conclude the first part of the thesis.

In the second part, we deal with named sets and HD-automata. In the light of simplifying and generalising the presentation, we introduce a structured definition of named sets very similar to that used in [FS06]: we introduce a category modelling name mappings in the presence of symmetry, which is then heavily used to define the product and power set.

A name abstraction functor for named sets is introduced first, and showed to be equivalent to the one over permutation algebras. The name abstraction functor is the categorical construction that is responsible, in \(	ext{Set}^I\), nominal sets, permutation algebras and named sets, of both name binding in the syntax and name allocation in the semantics of calculi. However, in named sets, due to locality of names, the definition of this functor is quite different from the other formalisms, and allows one to reuse previously generated fresh names in a correct way. This form of garbage collection is not only found in the final (abstract) semantics, but also in the system obtained from the operational rules, that is, the operational semantics. This is the distinguishing feature that allows named sets to be used as an implementation mechanism for nominal calculi (we will explain this in detail in §6.2), and motivates a “slogan” that summarises the point of view from which we develop our theory:

“named sets are nominal sets plus garbage collection”.

The resource being collected and reused here are the (no longer used) fresh names in the operational semantics of a given calculus. This is similar to the operation of the garbage collector in modern programming languages, that enables no longer used fresh locations to be reused to accommodate new allocation requests; as a consequence of the subsequent fragmentation, entire blocks of memory may be moved at each step, and to properly maintain references to memory, the correspondence between logical and physical locations is updated at each step of the computation by the means of an injective relocation; we will see how, in a similar fashion, history-dependent automata, as a special case of named functions, have a relabelling associated to each transition.

The above slogan has two intended meanings: on one hand, “named sets are nominal sets”. This was made clear by the equivalence results on the base categories of [FS06,
GMM06]. From a categorical point of view, there can not be an observable difference in two equivalent categories, as all universal properties are preserved and reflected by an equivalence. On the other hand, from the point of view of implementing algorithms on the semantics of a calculus, there is still a difference in employing a more compact representation, that gives rise to finite models in the operational semantics in many concrete cases.

The work on permutation algebras, and the results on the name abstraction functor in permutation algebras and named sets, have been published in [CM08].

Then, we proceed to define other functors commonly used in the semantics of name passing: product, coproduct and power set.

The corner stone of our development has been the definition of the categorical product in the category of named sets. This led to a formal understanding of locality of names: two elements of two different named sets can be related in many different ways, depending on how one binds their names together. Elements of the product are not just pairs, but there is a name mapping between the two related elements. This is a common situation in computing: whenever two extraneous entities are put in a relationship (e.g. two isolated networks that have to be interconnected), it is necessary to bind their common resources (e.g. addresses of nodes) establishing a correspondence on a case by case basis. It is surprising that this very general observation just comes from the standard construction of categorical product. The most important consequence is recovering history-dependent bisimulation and all the bisimulations on the various kinds of HD-automata as a standard definition: both the usual definition of bisimulation given in terms of a relation, and the one given in terms of kernels of morphisms into the final coalgebra, employ a subobject of the product. Thus, bisimilarity becomes a ternary relation, employing two states, and a name mapping between them. This is a common denominator among all the different versions of HD-automata, but it was not realized before that it is just the ordinary notion of bisimulation, with name mappings arising from the product.

The other half of the story is the definition of the countable power set. In [FMT05a], a normalisation step was employed in the action of the functor on arrows to solve the problem of redundant names that is introduced by the infinitary input action of the early $\pi$-calculus. We show that this corresponds to a finite representation of a subfunctor of the power set, thus proving the correctness of that approach.

The last part of the thesis deals with the final coalgebra theorem.
ence on the base categories to categories of coalgebras, we exploit pairs of equivalent functors. This can be, to a certain extent, compared to [BK05] and subsequent works, where pairs of dual functors are used to give an algebraic representation of categories of coalgebras. There, one is interested in finding a suitable category of algebras that provide a logical representation of a class of systems. In our case, the focus is on relating a category with pleasant algorithmic properties, that is, coalgebras over \textbf{NSet}, to categories having well-known mathematical properties (such as coalgebras over \textbf{FSAlg*}, or equivalently on “well behaved” presheaf categories like the Schanuel topos).

We keep as a running example a coalgebraic semantics of the early \(\pi\)-calculus. In particular, we will show how the framework we define can be used to translate this coalgebraic semantics to HD-automata in a modular way. The construction we derive corresponds to the definition of the functor in [FMT05a], and extends the correctness result to the infinite state case. Even though the finite state case is sufficient for minimisation, giving the full semantics of the calculus in terms of HD-automata is useful for various reasons, including a fully mathematical account of the minimisation algorithm whose correctness is now justified by existence of a final coalgebra, and the possibility to also implement “on-the-fly” strategies.

We remark that it is not sufficient, to recover a finite-state semantics of the early \(\pi\)-calculus, to exploit the categorical equivalence between named sets and sheaf models by direct translation of the semantics in [MP00] or [FT01]. As we explain, the mathematical counterpart of the early semantics of the \(\pi\)-calculus both in permutation algebras and in presheaf categories, is an infinite branching HD-automaton, due to the early input transitions. To recover finiteness of the semantics, additional work was done in [FMT05a] in the form of a normalisation step, originating from [FMP02]. We make sense of that construction in the coalgebraic setting, using accessibility results for functors that we define and their subfunctors.

Notice that the whole framework we define is aimed to represent the operational semantics of calculi. We do not provide tools to define languages, hence we are not concerned with issues like compositionality of definitions with respect to contexts, or finding rule formats that allow an easy definition of the semantics. These are completely orthogonal problems: on one hand, one can already define a compositional semantics, for example, in a presheaf category. On the other hand, various kind of rule formats already exist that allow to define such a semantics directly in presheaf or algebraic models. An
example using permutation algebras can be found in [BM02]. Such a semantics can then be used as a specification to define, by the equivalence results of [FS06] and [GMM06], a corresponding HD-automaton. However, without knowing how to translate the involved domains (in particular, the type of the coalgebra, that is, the involved functors), this is a non-trivial operation. This can be done in a modular way exploiting the functors we define in this work.

Map of the thesis

The thesis is organised as follows. We describe the necessary background in chapter 2, and in particular permutation algebras, the notion of theory morphism (which is later used to introduce an abstraction functor for permutation algebras), coalgebras, the relationship between accessibility of functors and final coalgebras, and the \( \pi \)-calculus which is used as a running example.

Chapter 3 is devoted to the definition of behavioural functors for permutation algebras (in particular name abstraction and its adjoint) and to explore their group-theoretical properties. Section 3.1 therein gives a number of results that are needed on permutation algebras. Some of these are well-known, most may be folklore, however there are exceptions, and we find it helpful to collect many relevant properties of the category in the same place. Chapter 4 contains two examples: one illustrating how De Bruijn indexes arise from the abstraction functor we define, and one showing how the semantics \( \pi \)-calculus is defined in permutation algebras.

In the second part of the thesis we introduce named sets and HD-automata: in chapter 5 we give the main definitions. The definition of the functors that are the main objective of the thesis is given in chapter 6, and the final coalgebra theorem in chapter 7. Two examples are finally provided in chapter 8: how to turn causal bisimulation for place-transition Petri nets of [BDKP91] into a coalgebraic definition exploiting the categorical product of named sets, and how to modularly define a functor for the early semantics \( \pi \)-calculus in the spirit of [FMT05a], in particular obtaining a finitely branching transition system apt to automated verification.
Motivation

At the beginning of my Ph.D. studies, I focused my research activity on verification methods for nominal calculi. In particular, I considered HD-automata that had been developed in Marco Pistore’s thesis [Pis99], and the possibility to reuse results from Emilio Tuosto’s thesis [Tuo03] in the context of model checking. To start with, I dissected the definitions that were previously given for the π-calculus and other formalisms. Then, I tried to explain my preliminary results on model checking to other people. It proved incredibly difficult to transfer the intuition behind any definition given in terms of named sets, from a person who had longly worked on the topic, to the rest of the world. The work in Emilio Tuosto’s thesis was already advanced enough that any further extension of it required to raise the level of abstraction.

Therefore, I tried to develop a general notation to describe construction on named sets, and realised that a much simpler presentation could be obtained in terms of basic functors such as the categorical product and name abstraction. In the meantime, the equivalence results shown by Marcelo Fiore and Sam Staton, also included in Sam Staton’s thesis [Sta07], and by Fabio Gadducci, Marino Miculan and Ugo Montanari, shed light on the link between presheaf models and HD-automata, and clarified resemblances and differences between the two.

Thus, the study of the basic constructions over nominal sets, that is, name abstraction, polynomial functors and power set, grew up and has become my thesis. Immediate achievements are characterising the definitions in Emilio Tuosto’s thesis in a compositional manner, and seeing that history-preserving bisimulation has a purely coalgebraic definition, thus formalising the intuition that to compare two systems with local resources, one first has to decide a mapping that binds their resources together. Proofs are generally easier than one might expect, coming from the equivalence of the involved categories, and accessibility of defined functors. This is in my opinion an evidence that the level of abstraction of this presentation is a reasonable choice.

A typical question that is made about named sets and HD-automata is what does one gain in the study of yet another model of nominal calculi, given that presheaf categories typically suffice to give a fully abstract semantics of such formalisms. An informal answer may be given starting from the observation that practical applications in computing are nowadays in bad shape, due to a widely increased complexity in the problems that are addressed. We are surrounded by, and use every day, software systems that we can not
rely on. This is likely the most important reason why formal methods are becoming so popular, in an attempt to automatise (part of) the work of a system designer. However, real world is quite complex, hence mathematical models of it are complex in turn. To the aim of developing formal methods in a rigorous way, algorithms that deal with these models should be directly derived from their mathematical descriptions. The complexity of a concrete and by hand implementation of a mathematical model may introduce subtle errors or overapproximations that would lead to unexpected results. In other words we need to “automatise automatisation” and obtain correct by design algorithms.

A perfectly explanatory example is iteration along the terminal sequence, the generalised minimization algorithm for systems described as coalgebras. The definition in the category of sets coincides with the Kanellakis-Smolka partition refinement algorithm for labelled transition systems. The definition on HD-automata for the π-calculus, developed in Marco Pistore’s works and Emilio Tuosto’s thesis, as a side effect, computes the greatest symmetry group of a system up-to bisimulation, a noticeable achievement that until now seems to have received less consideration than it deserves. Both algorithms are obtained from a simple and elegant construction, described by a few lines of mathematics.

In view of giving other similar results (I expect a model-checking algorithm to be the immediately following step), it is necessary to maintain a bridge between mathematical models and algorithms. The situation is particularly complex for calculi with name generation. Most of the existing verification tools avoid modelling fresh resource generation. It may be an intuitive choice to trace the history of resources along computation steps mimicking garbage collection in programming language semantics. As it has been stressed many times, subtle issues may arise when doing this. Symmetry of systems immediately come to mind.

An important problem is that dynamic allocation of resources leads to infinite state systems. This is addressed in this thesis defining a name abstraction functor for named sets. Another issue is that, when putting together different systems, concerns arise on how to compose their local histories. A proper way to do it comes with the definition of the categorical product of named sets that we give here.

To get back to the question, a good way to bridge nominal models of computation and algorithms, for what said above, can be the category of named sets. In the past, the inherent difficulty of describing a system in terms of this category and its coalgebras,
without having a compositional language that provides the necessary types, has made this powerful tool only available to the “initiated”, since a deep knowledge of the model is required each time a new language has to be mapped to HD-automata.

Therefore, I expect that the contribution that I present here can be a good toolkit and an operating manual to develop correct by construction formal methods for nominal calculi, taking benefit from the many theoretical results that are well known in more abstract models such as presheaf categories, and that this work can leverage the complexity of designing verification tools for nominal models.
Chapter 2

Background

2.1 Nominal Sets as Algebras for the Permutation Signature

Here we introduce the theory of sets equipped with a notion of action of a permutation over the set of natural numbers $\omega$. These are usually known as FM-sets, from the set theory of Fraenkel and Mostowski, who developed this model to show independence of the axiom of choice from other axioms of set theory. Their full subcategory of nominal sets was employed by Gabbay and Pitts to model name binding in [GP99] and subsequent works.

This category can be conveniently defined observing that FM-sets are algebras for the finite-kernel permutation group viewed as a monadic, single-sorted algebraic theory. These algebras have been referred to as permutation algebras and were employed, independently from the developments of nominal syntax, to model the semantics of name passing calculi [BM02, MP05], even involving name fusions [BM06]. The main advantage in using the algebraic definition is to obtain equivariant functions as the standard notion of algebra homomorphism. Also, one can reuse powerful tools from the well established theory of algebras (for example, theory morphisms used in §3.2 to define name abstraction). Finally, this also allows one to view nominal sets as a subcategory of the functor category $\mathbf{Set}^{\text{Aut}_\omega}$, indexed by the groupoidal category of permutations over $\omega$.

Here we give the basic definitions, among which the most important are those of support, symmetry and orbits. Together, these fundamental notions define permutation algebras up-to isomorphism.
CHAPTER 2. BACKGROUND

The set of permutations over a set $S$ forms a group where the operation is function composition. The core of the theory we present is the subgroup of finite-kernel permutations over the set of natural numbers $\omega$, which we will denote with $Autf$, without making an explicit distinction between it seen as a set, a group, a groupoidal category, or the algebraic specification in definition 2 below.

**Definition 1** (finite-kernel permutation). The kernel $\ker(\pi)$ of a permutation $\pi : \omega \to \omega$ is the set

$$\{ x \in \omega \mid \pi(x) \neq x \}$$

A permutation is finite-kernel if its kernel is finite. The set of all finite-kernel permutations over $\omega$ is denoted with $Autf$ and forms a group where the operation is function composition.

Now we introduce the notions of permutation signature and permutation algebra. With “permutation signature” we will actually refer to an algebraic specification made up of operations and equations, not just to a signature. However, we prefer to adhere to the notation that was used in previous works.

**Definition 2** (permutation signature). The (finite-kernel) permutation signature over $\omega$ is the set of finite kernel permutations over $\omega$, considered as unary, one-sorted algebraic operations, that is, the set:

$$\{ \pi : 1 \to 1 \mid \pi \in Autf \}$$

together with the equational axioms

$$id(x) = x$$

$$\pi_1(\pi_2(x)) = (\pi_1 \circ \pi_2)(x)$$

for each $\pi_1$ and $\pi_2$ in $Autf$.

**Definition 3** (permutation algebra). A permutation algebra is an algebra

$$A = \langle A, \{ \pi_A : A \to A \mid \pi \in Autf \} \rangle$$

for $Autf$, where $A$ is the carrier set, and $\pi_A$ is the interpretation of $\pi$, also called permutation action.

The notion of permutation algebra morphism is defined as follows.
2.1. NOMINAL SETS AS ALGEBRAS FOR THE PERMUTATION SIGNATURE

Definition 4 (morphism). An algebra morphism, between permutation algebras $A = \langle A, \{\pi_A\} \rangle$ and $B = \langle B, \{\pi_B\} \rangle$ is a function $f : A \rightarrow B$ which preserves operations of the permutation signature, i.e.

$$\forall x \in A. \forall \pi \in \text{Aut } f(\pi_A(x)) = \pi_B(f(x))$$

Such morphisms are called equivariant functions in the theory of Gabbay and Pitts. Notice that $\omega$ can be considered a permutation algebra using the natural interpretation $\pi_\omega = \pi$. We now introduce the group-theoretical notion of symmetry (also called isotropy group or stabiliser) of an element of a permutation algebra, and the definition of support and orbit.

Definition 5 (symmetry). Given a permutation algebra $A = \langle A, \{\pi_A\} \rangle$, the symmetry of an element $a \in A$ is the set of all permutations fixing $a$ in $A$, defined as $G_A(a) = \{ \pi \in \text{Aut}_A | \pi_A(a) = a \}$.

Definition 6 (identity group). The identity group of $X \subseteq \omega$ is defined as

$$\text{fix}(X) = \{ \pi \in \text{Aut}_A | \pi|_X = \text{id}_X \}$$

Definition 7 (support). We say that $X \subseteq \omega$ supports $a \in A$ if

$$\text{fix}(X) \subseteq G_A(a)$$

that is, all permutations fixing $X$ also fix $a$ in $A$. The least finite set $X$ satisfying this condition, if it exists, is called the support of $a$, written $\text{supp}_A(a)$. A permutation algebra is said to be finitely supported if all of its elements have finite support.

Each element of a permutation algebra is trivially supported by $\omega$. A finite supporting set might not exist, but if there is one, then the support is the intersection of all of them. The notion of support generalises that of “free variables” of terms, thus we will often refer to $\omega$ as the set of names. On the other hand, the notion of symmetry models indistinguishability of free names with respect to certain permutations.

Definition 8 (category of permutation algebras). Permutation algebras and their morphisms form a category, named $\text{Alg}^\pi$. We will denote with $\text{FSAlg}^\pi$ the full subcategory of finitely supported permutation algebras and their morphisms.

Finitely-supported permutation algebras are the nominal sets of Gabbay and Pitts. The last notion we need to introduce on permutation algebras is that of orbit.
Definition 9 (orbit). The orbit of \( a \in A \) is \( \text{orb}_A(a) = \{\pi_A(a) \mid \pi \in \text{Aut}_f\} \).

Orbits partition algebras in equivalence classes. We denote with \( a^o_A \) the canonical representative of the equivalence class of \( a \), and with \( X^o_A \) the set \( \{x^o_A \mid x \in X\} \), for \( X \subseteq A \). Orbits play a central role when switching from the category of permutation algebras to their “finitistic” counterpart, named sets.

Example 10. (terms with variables) Let \( \Sigma \) be a signature. Terms with variables in \( \omega \) form a permutation algebra

\[ T = \langle T_\Sigma(\omega), \{\pi_T\} \rangle \]

having permutation action

\[ \pi_T(t) = t[\pi(i)/i]_{i \in \omega} \]

(the substitution is actually on a finite number of names, since we employ finite-kernel permutations). It is easy to see that a finite set \( X \subseteq \omega \) supports a term \( t \) if and only if its set of variables \( \text{Var}(t) \) is a subset of \( X \). So, the least such set, i.e. the support, is the set of free variables of \( t \).

2.2 Other Presheaf Models and Equivalence Results

As we said, permutation algebras can be regarded to as the presheaf category \( \text{Set}^{\text{Aut}_f} \). Other presheaf models have been proposed for the syntax [FPT99] and the semantics [FT01] of name passing calculi. A comparison of the expressive power of various categories used to model name generation and passing, including nominal sets, the Schanuel topos and named sets, was done in [FS06] and [GMM06]. Here we give just a short introduction to the equivalence between finitely supported permutation algebras and the Schanuel topos, referring the reader to the papers we mentioned for further details.

Definition 11 (equivalence of categories). An equivalence \( \langle F, G, \eta, \epsilon \rangle \) between two categories \( A \) and \( B \) is made up of a pair of functors \( F : A \rightarrow B \) and \( G : B \rightarrow A \), and two natural isomorphisms \( \eta : \text{Id}_A \rightarrow G \circ F \) and \( \epsilon : F \circ G \rightarrow \text{Id}_B \). Two categories are said to be equivalent if there exists an equivalence between them.

Equivalences preserve and reflect many categorical properties, in particular, all limits and colimits. The category \( \text{FSAlg}^\pi \) is equivalent to the full subcategory of \( \text{Set}^I \) consisting of pullback preserving functors, that is, the Schanuel topos \( \text{Sh}(I) \). Observe that a
functor that preserves pullbacks also preserves monos. Thus, pullback preservation ensures that given a functor $F : I \to \text{Set}$, an element $x \in F(i)$ is uniquely represented by an element of $F(j)$ whenever $i \subseteq j$. Going further, given an element $x$ such that $x \in F(j)$ and $x \in F(k)$, mono preservation ensures that $x$ is uniquely represented in $F(j \cup k)$. Pullback preservation now implies $x \in F(j) \cap F(k)$. Thus, if $x \in F(i)$ for some finite set $i$, we can calculate its support which is the least finite set $s$ such that $x \in F(s)$. Pullback preservation in $\text{Set}^I$ corresponds to existence of the minimal support in $\text{Set}^{Autf}$.

The most important point about this particular equivalence result is that, whenever we have a semantics expressed in one of two equivalent formalisms, we can translate the semantics in the other one, and the most natural category for each specific purpose can be used.

An advantage of using staged presheaves is the simpler definition of the name abstraction functor, which is just $\delta(F)(i) = F(i + 1)$, and makes immediately clear how the fresh name is obtained (compare this with definition in [GP99], at the beginning of Sec. 4, and with the one that we give in §3.2). On the other hand, nominal sets are a bit closer to ordinary practice in programming language semantics: typically, one defines a countable set of names and a substitution operation. Using nominal sets, transition rules strongly resemble those in $\text{Set}$, with the addition of a binding operator that is easily defined using name abstraction (see §4.2).

These models are difficult to be implemented because of absence of garbage collection. We will see in this work how this problem is solved by named sets and the abstraction functor we define in §6.1. Equivalence results ensure that one can define the semantics of a language in a presheaf category, where it simpler in many cases, and then translate the results into nominal sets, to take advantage of the finitistic account of name generation.

### 2.3 Theory Morphisms

Theory morphisms, or views, are equation-preserving signature morphisms $M : \Sigma_1 \to \Sigma_2$, that yield algebras of $\Sigma_1$ from algebras of $\Sigma_2$. Here we just deal with the single-sorted case, since it is sufficient to present our work. The contents of this section are standard material from the theory of algebras (see e.g. [GB92]).

We denote with $T_\Sigma$ the initial algebra of the signature $\Sigma$, with $T_\Sigma(V)$ the free $\Sigma$-algebra over a set of variables $V$, and with $T_{\Sigma,E}(V)$ the free $\Sigma$-algebra over $V$ quotiented with equations derivable from $E$. The operations of the two initial algebras are indicated...
with $\text{op}_{T_{\Sigma}(V)}$ and $\text{op}_{T_{\Sigma,E}(V)}$, for $\text{op} \in \Sigma$. Given a $\Sigma$-algebra $A = \langle A, \{\text{op}_A | \text{op} \in \Sigma\} \rangle$, we call presentation of $A$, denoted with $\text{Pres}(A)$, the kernel of the unique $m : T_{\Sigma} \to A$, i.e. the set of pairs $t_1 = t_2$ such that $t_1, t_2 \in T_{\Sigma}$ and $m(t_1) = m(t_2)$. $\text{Eq}(E)$ represents the set of all equations derivable from $E$.

**Definition 12** (theory morphism). A signature morphism $M$ between signatures $\Sigma_1$ and $\Sigma_2$ is a function from the operators of $\Sigma_1$ to the operators of $\Sigma_2$ that respect operator arity, i.e. for every operator $\text{op}$ of arity $k$, $M(\text{op})$ has arity $k$. A signature morphism is inductively extended to $T_{\Sigma}(V)$ as

$$M(\text{op}(T_1, \ldots, T_k)) = M(\text{op})(M(T_1), \ldots, M(T_k))$$

(2.1)

and to equations as

$$M(T_1 = T_2) = (M(T_1) = M(T_2))$$

(2.2)

Given two specifications $S_1 = \langle \Sigma_1, E_1 \rangle$ and $S_2 = \langle \Sigma_2, E_2 \rangle$, a theory morphism from $S_1$ to $S_2$ is a signature morphism from $\Sigma_1$ to $\Sigma_2$ that preserves equations derivable from $E_1$, i.e.

$$(T_1 = T_2 \in \text{Eq}(E_1)) \implies (M(T_1 = T_2) \in \text{Eq}(E_2))$$

Every theory morphism induces a (forgetful) functor from the category of algebras of its destination to the category of algebras of its source, and it has a left adjoint.

**Definition 13** (forgetful functor). Let $Th_1 = \langle \Sigma_1, E_1 \rangle$ and $Th_2 = \langle \Sigma_2, E_2 \rangle$ be two specifications. A theory morphism $M : Th_1 \to Th_2$ associates to every $(\Sigma_2, E_2)$-algebra $A = \langle A, \{\text{op}_A | \text{op} \in \Sigma_2\} \rangle$ a $(\Sigma_1, E_1)$-algebra $U(A) = \langle A, \{\text{op}_U(A) | \text{op} \in \Sigma_1\} \rangle$ with the same carrier, where

$$\text{op}_U(A) = M(\text{op})_A$$

for each operator $\text{op}$ in $\Sigma_1$. The map $U$ extends to a functor, acting on arrows as

$$U(f) = f$$

$U$ has a left adjoint.

A definition of the left adjoint $F$ of $U$ can be given as a free construction that returns, for each $\Sigma_1$-algebra $A$, the free $\Sigma_2$-algebra over its carrier $A$, quotiented with the translation of $\text{Pres}(A)$ via $M$. 
Definition 14 (free functor). A theory morphism $M : \langle \Sigma_1, E_1 \rangle \rightarrow \langle \Sigma_2, E_2 \rangle$ associates to every $\langle \Sigma_1, E_1 \rangle$-algebra $A = \langle A, \{\text{op}_A\} \rangle$ a $\langle \Sigma_2, E_2 \rangle$-algebra $F(A) = \langle T_{\Sigma_2}(A)/E_2 \cup M(\text{Pres}(A)), \{[\text{op}_{T\Sigma_2}(A)]\} \rangle$

where the quotiented operation $[\text{op}_{T\Sigma_2}(A)]$ is defined as $[\text{op}_{T\Sigma_2}(A)]([a]) = [\text{op}_{T\Sigma_2}(A)(a)]$

This map extends to a functor, acting on arrows $f : \langle A, \{\text{op}_A\} \rangle \rightarrow \langle B, \{\text{op}_B\} \rangle$ as $F(f)([x \in A]) = [f(x)]$, $F(f)([\text{op}(T_1, \ldots, T_n)]) = [\text{op}(F(f)(T_1), \ldots, F(f)(T_n))]$.

Theorem 15. $F$ is left adjoint to $U$.

2.4 Multi-colimits

In [Die79] a weakened form of categorical limit is defined, where the limiting cocone is not unique. This construction, called multi-limit, obtains families of universal objects and morphisms, each one distinct from, and incompatible with, the other ones, recovering universality properties in categories without certain limits. In this work, we are interested in multi-colimits, whose definition (dual to that of multi-limits) is given below.

From now on, we denote with $|C|$ the objects of a category $C$, with $|D|$ the objects of a diagram $D$, and with $\text{hom}(c_1, c_2)$ the homset of $c_1$ and $c_2$. Moreover, we denote with $\langle t_i \rangle_{i \in I}$ a tuple indexed by a set $I$, omitting the range subscript when it is clear, i.e. just denoting a tuple as $\langle t_i \rangle$.

Definition 16 (multi-colimit). Given a diagram $D$ in a category $C$, the multi-colimit of $D$ is a set $\text{MCL}(D)$ of cocones over $D$ such that for all cocones $L' = \langle f_i : o_i \rightarrow o'_i \rangle_{i \in |D|}$ over $D$ there exists a unique cocone $L = \langle f_i : o_i \rightarrow o'_i \rangle_{i \in |D|} \in \text{MCL}(D)$ and a unique arrow $u : o \rightarrow o'$ making the diagram $L \cup L' \cup u$ commute.

When ordinary colimits exist, $\text{MCL}(D)$ is just a singleton containing the unique colimit. In this work, multi-colimits over diagrams of objects, i.e., multi-coproducts, are used as canonical representatives of cospans, to denote minimal (in the sense of existence of a unique mediating morphism) representatives of sets of cocones that are equivalent (in the sense that they map things in the same way into an intermediate object, whose precise identity does not matter).
2.5 Coalgebras, Bisimulation and Accessibility

Coalgebras (for an introduction, see [JR97, Rut00, Ada05]) are a categorical mathematical model that allows one to reason in terms of observable, extensional properties of a system, instead than using intensional properties such as physical equality. The theory of coalgebras is dual to the theory of algebras. The latter has been traditionally used to model data, while coalgebras typically model systems that have a possibly infinite evolution.

Definition 17 (coalgebra). Given a category $\mathbf{C}$ and an endofunctor $T : \mathbf{C} \to \mathbf{C}$ a coalgebra for $T$, or $T$-coalgebra, is an arrow $f : a \to T(a)$. Given two coalgebras $f : a \to T(a)$ and $g : b \to T(b)$, a coalgebra morphism is an arrow $h : a \to b$ such that

$$T(h) \circ f = g \circ h$$

For each category $\mathbf{C}$ and endofunctor $T$, $T$-coalgebras and their morphisms (illustrated in Fig. 2.1) form the category $\text{Coalg}(T)$.

An LTS is a function $f : a \to \mathcal{P}_{\text{fin}}(\mathcal{L} \times a)$ where $a$ is a set of states and $\mathcal{L}$ is set of labels. By viewing $\mathcal{P}_{\text{fin}}(\mathcal{L} \times -)$ as a functor, $f$ becomes a coalgebra in the category $\text{Set}$. The most important notion related to coalgebras is bisimulation, corresponding to an idea of equivalence where we do not compare systems by physical equality but by the observations we can make on its evolution. The standard definition of bisimulation is based on the notion of relation, that is, a subobject of the categorical product.

Definition 18 (bisimulation). Given two $T$-coalgebras $f : a \to T(a)$ and $g : b \to T(b)$, a bisimulation between them is a subobject $r$ of the categorical product, together with a span $h : r \to a, k : r \to b$ for which there exists a coalgebra $l : r \to T(r)$ making $h$ and $k$ coalgebra morphisms from $l$ to $f$ and $g$, respectively. The greatest bisimulation is called bisimilarity.
The definition is illustrated by the commutative diagram of Fig. 2.2.

In the case of LTSs, the definitions above specialise to the ordinary ones. The category \( \text{Coalg}(T) \) does not necessarily have a final object. An interesting case is when \( C \) is \textit{locally presentable} and \( T \) is \textit{accessible} (see [Ada05] for details). In this case, not only a final object exists, but since accessibility is in turn preserved by composition of functors, one is able to give modular definitions of categories of coalgebras having final objects, by defining a set of "basic" functors.

Having a final coalgebra is important because this gives a unique characterisation to the coalgebraic semantics of a given programming language, or more generally because elements of the final coalgebra represent abstract semantics, just like elements of the initial algebra represent abstract syntax.

Finally, existence of the final coalgebra ensures that there is a unique (minimal) system for each class of bisimilar ones. This is exploited in a generalisation of the partition refinement algorithm, known as \textit{iteration along the terminal sequence} (which is studied in detail in [Wor99]). This algorithm may terminate even if there is no final coalgebra. In this case, however, the minimal system is \textit{not} the canonical representative of all bisimilar states, but rather different systems may be obtained, starting from different, but bisimilar, systems. These different systems can no longer be minimised and are bisimilar to each other.

If a final object exists in \( \text{Coalg}(T) \), \( C \) has pullbacks, and \( T \) weakly preserves pullbacks, then the following definition of bisimulation coincides with definition 18.

\textbf{Definition 19} (bisimulation as a kernel). A bisimulation is the kernel of a \( T \)-coalgebra morphism \( f \), i.e. the pullback object of the pair \((f, f)\). The kernel of the unique morphism into the final coalgebra is called bisimilarity.

In many categories (e.g. in \( \text{Set} \) and \( \text{FSAlg}^\pi \)), the kernel of a morphism may be seen as a quotient over bisimilar states. In this light, bisimilarity is the greatest such quotient, which corresponds to the ordinary definition over LTSs.
An even more general definition of observational equivalence, not requiring products, is *behavioural equivalence* as defined in [Kur00], using epi cospans instead of strong mono spans. All the definitions coincide when the base category is a topos and the functor weakly preserves pullbacks.

### 2.6 Final Coalgebras for Accessible Endofunctors

In this work we employ the class of *accessible* functors as a mathematical tool to obtain a compositional class of endofunctors admitting a final coalgebra. Here we recall the needed definitions and the theorem that we will use, directly taken from [Wor99]. Coalgebras and accessibility are extensively dealt with in [Wor99, Ada05, AR94].

**Definition 20** (locally presentable category). Let $\lambda$ denote an infinite regular cardinal. An object $o$ in a category $C$ is $\lambda$-presentable if its homset functor $\text{hom}(o, -)$ preserves $\lambda$-filtered colimits. A category $C$ is locally $\lambda$-presentable if it is cocomplete and there is a set $A$ of $\lambda$-presentable objects such that every object is a $\lambda$-filtered colimit of objects from $A$. $C$ is locally presentable if it is locally $\lambda$-presentable for some $\lambda$.

**Definition 21** (accessible functor). Let $C_1$ and $C_2$ be locally $\lambda$-presentable categories. A functor $T : C_1 \to C_2$ is $\lambda$-accessible if it preserves $\lambda$-filtered colimits, and it is accessible if it is $\lambda$-accessible for some $\lambda$.

It is well known (see e.g. [AR94]) that the Schanuel topos is accessible. Since equivalences preserve accessibility, also named sets and permutation algebras are accessible. Summarising the results of [Wor99] and related works, we state the following theorem.

**Theorem 22.** Any accessible endofunctor on a locally presentable category admits a final coalgebra. If the functor preserves monos, then the terminal sequence converges to the final coalgebra.

Indeed, convergence to the final coalgebra is not related to termination of the minimisation procedure, however it ensures that, if the algorithm terminates, it yields the same model when run on two bisimilar systems, that is, it returns canonical representatives of sets of bisimilar systems.

Accessibility and preservation of monos are preserved by composition of functors; this makes all results related to accessibility and mono preservation inherently compositional. For the functors that have a counterpart in $\textbf{Set}$, it is easy to show accessibility.
2.7. The $\pi$-calculus

The $\pi$-calculus was presented by Milner, Parrow and Walker [MPW92]. It is a process algebra featuring interleaving concurrency and nondeterminism, and being a nominal calculus it has name generation, name passing and a notion of equality over names. The introduction of name passing gave to the language the power to dynamically reconfigure the communication structure. Since its introduction, the calculus has been object of intensive study, both to test its expressive power and limitations, and to exploit practical possibilities of formal methods applied to mobility. The combination of name generation and passing can express a great deal of programming language features, ranging from unique objects to function and method calls.

2.7.1 Syntax

Given a denumerable infinite set of names $\mathcal{R} = \{x_0, x_1, x_2, \ldots\}$, the set of $\pi$-calculus agents is defined by the syntax

$$P ::= 0 \mid \alpha P \mid P_1 \parallel P_2 \mid P_1 + P_2 \mid (\nu x) P \mid [x = y]P \mid A(x_1, \ldots, x_{r(A)})$$

$$\alpha ::= \tau \mid x(y) \mid \overline{x}y$$
(alpha) \( P \equiv Q \) if \( P \) and \( Q \) are alpha equivalent

(sum) \( P+0 \equiv P \), \( P+Q \equiv Q+P \), \( P+(Q+R) \equiv (P+Q)+R \)

(par) \( P | 0 \equiv P \), \( P | Q \equiv Q | P \), \( P | (Q | R) \equiv (P | Q) | R \)

(res) \( (\nu x) \ 0 \equiv 0 \), \( (\nu x) \ (\nu y) \ P \equiv (\nu y) \ (\nu x) \ P \), \( (\nu x) \ (P | Q) \equiv P | (\nu x) \ Q \) \( \text{if } x \notin fn(P) \)

(match) \( [x=y]P \equiv [y=x]P \)

Figure 2.3: Structural congruence

where \( r(A) \) is the rank of the process identifier \( A \). Free names and bound names of an agent \( P \) are defined as usual and indicated with \( fn(P) \) and \( bn(P) \), with \( n(P) = fn(P) \cup bn(P) \); name \( y \) is bound in \( x(y).P \) and \( (\nu y) \ P \).

Each identifier \( A \) has a definition \( A(y_1, \ldots , y_{r(A)}) = P_A \) (with \( y_i \) all distinct and \( fn(P_A) \subseteq \{ y_1 \ldots y_{r(A)} \} \) ) and each identifier in \( P_A \) is in the scope of a prefix (guarded recursion). The algebra of terms is quotiented by structural congruence, whose axioms are reported in Fig. 2.3.

A quick review of the syntax will serve us to informally explain the calculus:

- the inactive process is called \( 0 \) and does nothing;
- the syntactical category \( \alpha \) is that of the so-called prefixes; \( \tau \) corresponds to internal computation, \( x(y) \) is the synchronous input prefix, binding name \( y \) to the received channel, \( \bar{xy} \) is the synchronous output prefix, sending \( y \) over the channel \( x \);
- the process \( \alpha.P \) performs the action corresponding to \( \alpha \) and then behaves like \( P \);
- the construct + is nondeterministic choice
- the construct | is parallel (interleaving) composition
- \( [x=y]P \) is the matching construct: the obtained process can behave like \( P \), but only when \( x \) and \( y \) are equal;
- \( (\nu x) \ P \) is the restriction operator: it makes the name \( x \) a local name that is unknown to any other process, until it is transmitted over a common channel;
- finally, an iterative behaviour can be expressed using recursive definitions taking names as arguments.
2.7. THE π-CALCULUS

2.7.2 Early Operational Semantics and Bisimulation

The early semantics of the π-calculus was given in [MPW92] as a labelled transition system. However, as we will see, the definition of bisimulation is not coalgebraic, due to fresh name generation. The LTS semantics is defined by the inference rules given in Fig. 2.4, augmented with a rule that takes structural congruence into account:

\[ P \equiv Q, Q \xrightarrow{\alpha} Q', Q' \equiv P' \]

Notice the three side conditions on freshness of names in the rules. We can now introduce the definition of early bisimulation for the π-calculus:

**Definition 24** (early bisimulation). A relation \( R \) over agents is an early simulation if whenever \( P \xrightarrow{R} Q \) then:

- for each \( P \xrightarrow{\alpha} P' \) with \( bn(\alpha) \cap fn(P, Q) = \emptyset \) there is some \( Q \xrightarrow{\alpha} Q' \) such that \( P' \xrightarrow{R} Q' \).

The side condition ensures that there is no free name capture in both processes. Notice that the label is \( \alpha \)-convertible and free names of both the agents are in a finite number. A relation \( R \) is an early bisimulation if both \( R \) and \( R^{-1} \) are early simulations.

Two agents \( P \) and \( Q \) are early bisimilar if \( P \xrightarrow{R} Q \) for some early bisimulation \( R \).
2.8 Permutation Algebras for the $\pi$-calculus

Early bisimulation for the $\pi$-calculus of definition 24 is not coalgebraic, due to the side condition on freshness of names. To rectify this, the solution is to change the base category to one that takes names into account. Early attempts in this direction include [FMS96], [Sta96] and [CSW97], where presheaf models were employed to represent name passing. These models were already implicitly coalgebraic, and explicit definitions of coalgebras over presheaf models where used in [FT01] for the semantics of the $\pi$-calculus. Even though it was recognised as a presheaf category only later, the category of finitely supported permutation algebras falls in this development line: in [MP00] and [MP05] we find a fully abstract coalgebraic model of the early $\pi$-calculus using coalgebras over permutation algebras. The technique employed there, presented in turn in [CHM99], is to lift SOS rules from the category of sets to so-called structured coalgebras, that is, bialgebras over Set where the algebraic theory associated to the functor may have axioms. The obtained semantics is a bialgebra, and can be viewed as a coalgebra over $\text{FSAlg}_\pi$. In that case, bisimulation is a congruence with respect to finite-kernel permutations.

Handling of fresh names is obtained using a shift operator $(-)^{+1} : \text{Autf} \to \text{Autf}$ that is used to introduce a fresh name 0. Transitions performing a bound output shift the names of the reached states by one, making room for a fresh name 0. Since the bound name is always 0, there is no need to indicate it in the transition label. This machinery, similar to De Bruijn indexes [dB72], allows to obtain a fully abstract model, but its use was never formalised. This is indeed the aim of §4.2, where we show that this operation is due to the notion of name abstraction on $\text{FSAlg}_\pi$ that we define, extending the name shift operator to a functor induced by a theory morphism.

**Definition 25** (right shift). The right shift operator $-^{+1} : \text{Autf} \to \text{Autf}$ gives, for each permutation $\pi$, the operation $\pi^{+1}$ such that

$$\pi^{+1}(i) = \begin{cases} 
0 & \text{if } i = 0 \\
\pi(i) + 1 & \text{otherwise}
\end{cases}$$

The functor giving the signature of the coalgebra was not made explicit in [MP05] (we do this in §4.2 using the abstraction functor that we define). Briefly, the framework of [CHM99] prescribes to define a transition specification. This is an equational specification expressed as a set of “meta-rules” that specify the possible format of transitions.
Transition rules have to obey to this specification in order to obtain a bialgebra. The transition specification $\Delta_\pi$ for the $\pi$-calculus given in [MP05] can be found in Fig. 2.5. In this case, the algebraic operations are substitutions, and the proof obligation for actual transitions to respect the specification is that the action of a permutation on $\textit{bout}$ transitions shifts names in the destination.

The transition rules employ the permutation algebra $P_i$ of $\pi$-calculus agents, with permutation action defined using the ordinary notion of name substitution, and a permutation algebra of labels

$$\{\textit{tau}, \textit{in}(x,y), \textit{out}(x,y), \textit{bout}(x) \mid x, y \in \omega\}$$

with the usual permutation action. Labels $\textit{tau}, \textit{in}, \textit{out}$ have the meaning of synchronisation, input and output, while label $\textit{bout}$ corresponds to bound output, and features only one name.

The transition rules are defined in Fig. 2.6, and are based on the LTS rules of the $\pi$-calculus, using the “binding” permutation $\sigma^{i,p}$ that shifts all names in $\text{supp}_{P_i}(p)$ but $i$, which is sent in 0, defined as

$$\sigma^{i,p}(j) = \begin{cases} 
0 & \text{if } j = i \\
 j + 1 & \text{if } j \in \text{supp}_{P_i}(p) \setminus i \end{cases}$$

and completed in order to obtain a permutation (see remark 27).

With this definition, bisimulation becomes coalgebraic: there are no bound names at all in labels, and when bound output transition match, we are guaranteed that name 0 is the same in both reached states. At first sight, all this may look like an algorithmic
“trick” to allocate a new name. In the first part of this thesis we will carefully explain how this construction is due to a name abstraction functor that precisely corresponds to De Bruijn indexes when applied to the \( \lambda \)-calculus.
Part II

Permutation Algebras
Introduction

In this part of the thesis, we develop the theory of permutation algebras a step further by defining a name abstraction functor, and showing how this gives rise to De Bruijn indexes and to the functor for the semantics of the $\pi$-calculus given in [MP00]. The theory presented there was the starting point to define the minimisation algorithm for the $\pi$-calculus of [FMP02, FMT05a].

As it was soon recognised, even though they were developed independently, the category of nominal sets, and that of finitely supported permutation algebras, are essentially the same. We remark that the latter is not a new definition, but rather the standard notion of algebra for a specification. While the idea of modelling binding in this category certainly has its roots in the work on nominal syntax, the key role of the symmetry of elements, which subsumes the notion of finite support and its properties (see theorem 28 in the following) is a result of the research on permutation algebras. A key motivation for studying symmetries was the development of history-dependent automata with symmetries, that we deal with in the second part of the thesis.

A point of interest in pursuing the algebraic definition of nominal sets is that it obtains many constructions using standard algebraic techniques. As an example, we have seen that equivariant functions are just the algebra morphisms of the category. In §6.1 we will use the notion of theory morphism to obtain the name abstraction functor.

As we said, permutation algebras are the presheaf category $\mathbf{Set}^{\text{Aut}}$. The difference between this category and $\mathbf{Set}^\dagger$ is that the latter corresponds to a staged version. A reason to study in detail the untyped setting is that it is very close to ordinary set theory: to make use of the framework, one just has to define a permutation action, which is, to quote [MP00], “the smallest information required to define a semantically correct mechanism of name deallocation”. The permutation action is also sufficient, via the categorical equivalence of [GMM06], to translate obtained results to history-dependent automata in order to implement algorithms that manipulate the syntax or the semantics of name passing.
Chapter 3

Behavioural Functors for Permutation Algebras

3.1 Some Results on Permutation Algebras

Here we survey a number of results that are needed in this part of the thesis. Many of these are folklore, but we find it very useful to put all of them together. Notice for example theorem 28, a simple result that explains in terms of symmetry one of the most pervasive properties of the support: it never grows along morphisms. This leads, in all categorical formalisms that handle names using injective relabelings (presheaves, nominal sets and named sets), to the necessity of defining specialised functors for name abstraction.

First of all, we show that the action of a permutation $\pi$ on elements of a finitely supported permutation algebra is determined by the action of $\pi$ on their support.

**Lemma 26.** In a finitely supported permutation algebra $A = \langle A, \{ \pi_A \} \rangle$, for each $a \in A$, we have $\pi|_{\text{supp}_A(a)} = \pi'|_{\text{supp}_A(a)} \implies \pi_A(a) = \pi'_A(a)$.

**Proof.** The permutation $\pi^{-1} \circ \pi'$ fixes $\text{supp}_A(a)$, thus by definition of support we have

$$(\pi^{-1} \circ \pi')_A(a) = a$$

Applying $\pi$ to both sides, we have $(\pi \circ \pi^{-1} \circ \pi')_A(a) = \pi(a)$, hence $\pi'_A(a) = \pi_A(a)$. □

**Remark 27.** Because of lemma 26, we will usually define a permutation $\pi$ only on the support of an element $a \in A$, when it is clear from the context that $\pi$ is to be applied only
to a. In this case, we assume that the definition of \( \pi \) is completed in order to obtain a finite-kernel permutation.

The following theorem asserts that the symmetry may grow, and does not shrink, along morphisms. In particular, its subsequent corollary gives an important property of nominal sets, extensively used in proofs about this category: the support never grows along morphisms. Even though the corollary is well known (see e.g. corollary 9 in [GMM06]), it is interesting to observe that it just comes from the symmetry of elements.

**Theorem 28.** Let \( f : \langle A, \{\pi_A\} \rangle \to \langle B, \{\pi_B\} \rangle \). Then

\[
\forall a \in A. G_A(a) \subseteq G_B(f(a))
\]

**Proof.** Let \( \pi \in G_A(a) \), then \( \pi_A(a) = a \). We have \( f(a) = f(\pi_A(a)) = \pi_B(f(a)) \), hence \( \pi_B \in G_B(f(a)) \).

**Corollary 29.** For each morphism \( f : A \to B \), we have

\[
\text{supp}_B(f(a)) \subseteq \text{supp}_A(a)
\]

**Proof.** We have \( \text{supp}_B(f(a)) \subseteq s \) for any \( s \) supporting \( f(a) \) in \( b \), i.e. such that \( \text{fix}(s) \subseteq G_B(f(a)) \). We have \( \text{fix}(\text{supp}_A(a)) \subseteq G_A(a) \). By theorem 28, \( G_A(a) \subseteq G_B(f(a)) \), hence \( \text{supp}_A(a) \) supports \( f(a) \) in \( B \), from which the thesis.

The following “isomorphism theorem” is of fundamental importance for named sets, since it asserts that a named set represents a class of isomorphic permutation algebras, as we will see in §5.

**Theorem 30.** Two permutation algebras \( A = \langle A, \{\pi_A\} \rangle \) and \( B = \langle B, \{\pi_B\} \rangle \) are isomorphic if and only if there exists a choice of canonical representatives of orbits, and an isomorphism \( i : A^\circ \to B^\circ \) in \( \text{Set} \), such that \( G_A(a^\circ) = G_B(i(a^\circ)) \).

**Proof.** For the “if” part, let \( f \) be defined as \( f(a^\circ) = i(a^\circ), f(\pi_A(a^\circ)) = \pi_B(i(a^\circ)) \). By definition, \( f \) is a permutation algebra morphism. Suppose that it is not injective. Then we have \( a_1 \neq a_2 \) and \( f(a_1) = f(a_2) \). We have that \( a_1 \) and \( a_2 \) belong to the same orbit: \( i \) is an isomorphism between canonical representatives, hence it can not send two elements of different orbits into the same element. Assume without loss of generality that \( a_1 \) is a canonical representative and let \( \rho \) be a permutation such that \( \rho_A(a_1) = a_2 \).
Then \( \rho_B(i(a_1)) = f(a_2) \) and \( \rho \) belongs to \( G_B(i(a_1)) \) which is a contradiction because \( \rho \notin G_A(a_1) \).

For the “only if” part, if \( f \) is a permutation algebra isomorphism, with \( f^{-1} \) its inverse, we can apply theorem 28 in both directions and obtain equality of symmetries. Since morphisms respect orbits, we can chose \( f(a_o) \) as a canonical representative of its orbit.

Finally, we provide a “representation theorem”, taken from [MP05], aimed at giving a finite representation of the symmetry of finitely supported permutation algebras.

**Theorem 31.** The symmetry \( G_A(a) \) of \( a \in A \) is obtained by composition of two subgroups as follows:

\[
G_A(a) = \text{fix}(\text{supp}_A(a)) \circ (G_A(a) \cap \text{fix}(\omega \setminus \text{supp}_A(a)))
\]

**Proof.** By definition of support, \( \text{fix}(\text{supp}_A(a)) \subseteq G_A(a) \). Moreover, each permutation \( \pi \notin \text{fix}(\text{supp}_A(a)) \), such that \( \pi \in G_A(a) \), can be obtained as

\[
(\pi|_{\omega \setminus \text{supp}_A(a)} \cup \text{id}_{\text{supp}_A(a)}) \circ (\pi|_{\text{supp}_A(a)} \cup \text{id}_{\omega \setminus \text{supp}_A(a)}).
\]

The former is in \( \text{fix}(\text{supp}_A(a)) \), the latter in \( G_A(a) \cap \text{fix}(\omega \setminus \text{supp}_A(a)) \).

In words, the infinite set of all permutations in \( G_A(a) \) can be reconstructed from the information described by the (finite) set of all permutations in \( G_A(a) \) that only alter the support of \( a \), by composition with all the permutations that only alter names outside the support of \( a \). This theorem is exploited in named sets to obtain a finite description of the symmetries. Such a finite description is still a group, hence it can be efficiently described (using its generators, see [FMT05a]).

### 3.2 Name Abstraction and Concretion

Here we introduce a name abstraction functor for permutation algebras, and its left adjoint. Name abstraction is able to model binding and name allocation. The notions of abstraction and concretion that we define here are isomorphic to those defined in [GP99] and related works. We use a different definition (namely, employing a shift of all the names instead of a quotient on \( \alpha \)-equivalent terms) to recover exactly the semantics of the \( \pi \)-calculus defined in [MP00]. This sheds light on the technique of De Bruijn indexes, which is often considered as a technique to avoid a proper formalisation of
binding. In §4.1 we show that the abstract syntax of \( \lambda \)-calculus using De Bruijn indexes is obtained as an initial algebra for a specific functor in \( \text{FSA} \text{lg} \) involving our definition of name abstraction. Thus, this representation technique for binding is located at the same conceptual level of name abstraction of Gabbay and Pitts, and the corresponding notion in \( \text{Set} \). The nominal syntax of the \( \lambda \)-calculus obtained as the initial algebra of a functor involving name abstraction is exactly De Bruijn notation. In the semantics of nominal calculi, bisimilarity of coalgebras for name abstraction constrains fresh names to correspond on both sides along transitions. Using a basic meta-language for binding, we will see in §4.2 how the semantics of the \( \pi \)-calculus can just be represented using binding transitions (in a similar fashion to nominal inference rules of [Pit03]) in inference rules.

Name abstraction is the forgetful functor induced by a particular theory morphism, which embeds the set of permutations over \( \omega + 1 \) into the set of permutations over \( \omega \), exploiting the famous Hilbert’s hotel, i.e. shifting all the natural numbers by one, to “make room” for the fresh element 0. By viewing the model as a category of algebras, we can characterise abstraction, concretion and their adjunction as a standard result.

### 3.2.1 The Name Abstraction Functor

A theory morphism from \( \text{Autf} \) to itself is a function that preserves identity and composition. We define such a morphism, the right shift of a permutation \( \pi \) which we denote with \( \pi^+ \). The purpose of this operation is to let \( \pi^+ \) act like \( \pi \) but on names starting from 1 instead of 0. This way, name 0 is not changed by \( \pi^+ \) for any \( \pi \). Name 0 becomes fresh, and this choice has consequences on the support and symmetry of elements, as we will see. We recall the right shift operator of definition 25:

\[
\pi^+(i) = \begin{cases} 
0 & \text{if } i = 0 \\
\pi(i) + 1 & \text{otherwise}
\end{cases}
\]

Indeed, it is a theory morphism.

**Lemma 32.** The right shift operator is a theory morphism.

**Proof.** We have \( id^+(0) = 0 \) and \( id^+(i + 1) = id(i) + 1 = i + 1 \). Moreover

\[
\begin{align*}
(\pi \circ \pi')^+(0) &= 0 = (\pi^+ \circ (\pi')^+)(0) \\
(\pi^+ \circ (\pi')^+)(i) &= \pi^+((\pi')^+(i) + 1) = \pi((\pi')^+(i)) + 1 = (\pi \circ \pi')^+(i + 1)
\end{align*}
\]
Some more intuition is given by observing that \( \pi^{+1} \) is equal to \( \iota \circ [\pi; \text{id}_1] \circ \iota^{-1} \), where \( \iota \) is any isomorphism from \( \omega + 1 \) to \( \omega \) and the notation \([f; g]\) indicates the copairing of \( f \) and \( g \). In presheaf categories such as \( \textbf{Set}^I \) abstraction is defined on a presheaf \( T \) as \( \delta(T)(n) = T(n + 1) \), and on morphisms (that is, natural transformations) as \( \delta(f) = [f; \text{id}_1] \). Since we are working in the unsorted presheaf category \( \textbf{Set}^{Aut_f} \), we have to embed \( \omega + 1 \) into \( \omega \).

We have chosen the particular isomorphism \( \hat{\iota} \) sending \( \langle 0, * \rangle \) to 0 and \( \langle 1, i \rangle \) to \( i + 1 \) (the famous “Hilbert’s hotel”), but any other choice is possible (the choice we make matches the idea of De Bruijn indexes). We can describe all these different \( \iota \) as the set \( \{ \rho \circ \hat{\iota} \mid \rho \in \text{Autf} \} \). The effect of a generic \( \iota \) is to “make room” for a fresh name, that actually comes from the embedding of \( \omega \oplus 1 \) into \( \omega \).

We obtain the abstraction functor as the forgetful functor associated to the right shift theory morphism.

**Definition 33** (name abstraction in \( \text{FSAlg}^\pi \)). The endofunctor for name abstraction \( \delta : \text{Alg}^\pi \to \text{Alg}^\pi \) acts on objects as
\[
\delta(\langle A, \{\pi_A\} \rangle) = \langle A, \{\pi_A^{+1}\} \rangle
\]
and on arrows as
\[
\delta(f) = f
\]

The action of a permutation in \( \delta(A) \) cannot touch the name 0. The consequences of this are examined in \( \S 3.2.2 \).

**3.2.2 Properties of Abstraction**

In this section we study the support, symmetry and orbits of elements of finitely supported permutation algebras obtained using \( \delta \). In particular, we show that it restricts from \( \text{Alg}^\pi \) to \( \text{FSAlg}^\pi \), and how in \( \delta(A) \) we find more distinct orbits than in \( A \), containing the hidden elements of \( \delta(A) \), i.e. those that have a “bound” name, that is not observable.

**Theorem 34.** The support and symmetry of elements of \( \delta(A) \) are obtained as
\[
\text{supp}_{\delta(A)}(a) = \{i - 1 \mid i \in \text{supp}_A(a) \setminus 0\}
\]
\[
\mathcal{G}_{\delta(A)}(a) = \{\pi \mid \pi^{+1} \in \mathcal{G}_A(a)\}
\]
Proof. The result about the symmetry is trivial:

\[ \pi \in \mathcal{G}_{\delta(A)}(a) \iff \pi^{1+1}(a) = a \iff \pi^{+1} \in \mathcal{G}_A(a) \]

Now we show that \( S_a = \{ i - 1 \mid i \in supp_A(a) \setminus \{ 0 \} \} \) supports \( a \) in \( \delta(A) \):

\[ \pi \in fix(S_a) \]
\[ \implies \forall i \in supp_A(a) \setminus \{ 0 \}. \pi(i - 1) = i - 1 \]
\[ \implies \forall i \in supp_A(a). \pi^{+1}(i) = i \]
\[ \implies \pi^{+1} \in fix(supp_A(a)) \]
\[ \implies \pi^{+1} \in \mathcal{G}_A(a) \]
\[ \implies (by \ the \ result \ on \ the \ symmetry) \]
\[ \pi \in \mathcal{G}_{\delta(A)}(a) \]

Finally, suppose that \( X \not\subseteq S_a \) supports \( a \) in \( \delta_A \). Let \( Y = \{ 0 \} \cup \{ i + 1 \mid i \in X \} \), so that by definition of \( S_a \) we have \( supp_A(a) \not\subseteq Y \). We show that \( Y \) supports \( a \) in \( A \), thus getting to a contradiction:

\[ \bar{\pi} \in fix(Y) \]
\[ \implies (since \bar{\pi}(0) = 0) \]
\[ \exists \pi. \bar{\pi} = \pi^{+1} \land \pi^{+1} \in fix(Y) \]
\[ \implies (by \ hypothesis) \]
\[ \pi \in fix(X) \]
\[ \implies \pi \in \mathcal{G}_{\delta(A)}(a) \]
\[ \implies (by \ the \ result \ on \ \mathcal{G}_{\delta(A)}) \]
\[ \pi^{+1} \in \mathcal{G}_A(a) \]
\[ \implies \bar{\pi} \in \mathcal{G}_A(a) \]

\( \square \)

The above theorem proves that \( \delta \) restricts from \( \text{Alg}^\pi \) to \( \text{FSAlg}^\pi \). The intuition behind it is that, in \( \delta(A) \), we remove \( 0 \) from the support of each element. This way, name \( 0 \in supp_A(a) \) becomes fresh in \( \delta(A) \): no observation can be made about it, but it is still a hidden name of \( a \). This name can be used, exploiting the action of \( \delta \) on arrows: we just have \( \delta(f(a)) = f(a) \), hence \( f \) can use all the names of \( a \).
The property of 0 being *fresh* is also assured by the symmetry of \( a \) in \( \delta(A) \): \( G_{\delta(A)}(a) \) is the subgroup of \( G_{A}(a) \) that fixes 0, shifted by one name. The intuitive meaning of such property is that information about interchangeability of 0 is thrown away, making it distinct from any other name.

We now define a set of permutations used to describe orbits of \( \delta(A) \). Below, the finite set \( S \) will be used as the support of an element of a permutation algebra, hence by the convention of remark 27 we define these permutations only on \( S \), assuming that the definitions are appropriately completed to obtain a permutation.

**Definition 35** (binding permutations). Given a finite set \( S \), we define a permutation \( \pi_{\text{old}}^{(S)}(S) \) such that
\[
\pi_{\text{old}}^{(S)}(i) = i + 1 \text{ for } i \in S
\]
and \( |S| \) permutations \( \pi_{\text{h}}^{(S,i)} \), for \( i \in S \), such that
\[
\pi_{\text{h}}^{(S,i)}(j) = \begin{cases} 
0 & \text{if } j = i \\
1 & \text{if } j \in S \setminus i
\end{cases}
\]

Now we define functions in \( \text{Set} \) acting on carriers of permutation algebras. One is called *old*, because it embeds an element \( a \) from \( A \) into \( \delta(A) \) preserving all of its properties (support, symmetry, orbit). The other ones are called *hidden* since they obtain, from \( a \), new elements in \( \delta(A) \), whose properties can not be recovered in \( A \).

**Definition 36** (metalinguage of binding). The *old* element \( \text{old}_A(a) \) and the \( i \)th *hidden* element \( \text{hid}_i^A(a) \) of \( a \in A \) are defined as
\[
\text{old}_A(a) = \pi^{\text{old} \text{(supp}_A(a))}_A(a) \\
\text{hid}_i^A(a) = \pi^{\text{h} \text{(supp}_A(a),i)}_A(a)
\]

It is straightforward to check that *old* is a permutation algebra morphism of type \( A \to \delta(A) \). It holds that \( \text{supp}_{\delta(A)}(\text{old}_A(a)) = \text{supp}_A(a) \), \( G_{\delta(A)}(\text{old}_A(a)) = G_A(a) \), and \( \text{orb}_{\delta(A)}(\text{old}_A(a)) = \{ \text{old}_A(x) \mid x \in \text{orb}_A(a) \} \). In other words, *old* is an embedding of \( A \) in \( \delta(A) \).

The crucial property of *hid* \( i \) is to send name \( i \) to 0, hence we have (by theorem 34), \( \text{supp}_{\delta(A)}(\text{hid}_i^A(a)) = \text{supp}_A(a) \setminus i \). In words, for each element \( a \) and each name \( i \) of \( a \), we can identify an element of \( \delta(A) \) which has the same names as \( a \), minus \( i \). As we will see in §4.2, such an operation is of fundamental importance to define coalgebras for \( \delta \), allowing these to allocate fresh names along transitions.
Remark 37. In the following, we will use $\text{hid}_A(a)$ to define an element of $\delta(A)$: the subscript $A$ denotes the application of the permutation action in $A$, thus identifying an element of the carrier $A$, which is also the carrier of $\delta(A)$, not the fact that element $\text{hid}_A(a)$ belongs to permutation algebra $A$ as one might expect.

We show that the old and hidden elements form a partition of $\delta(A)$.

Lemma 38. For each $a \in A$, there exist either $b \in A$ such that $a = \text{old}_A(b)$, or $b \in A$ and $i \in \text{supp}(b)$ such that $a = \text{hid}_A^i(b)$.

Proof. We define a permutation $\pi$ as follows: if $0 \in \text{supp}(A(a))$, we put $\pi(0) = i$, with $i \notin \text{supp}(A(a))$, and $\pi(j) = j - 1$ for $j \in \text{supp}(A(a)) \setminus 0$; if $0 \notin \text{supp}(A(a))$, we just put $\pi(j) = j - 1$ for $j \in \text{supp}(A(a))$. In the first case, we have $a = \text{hid}_A^i(\pi(a))$, while in the second case we have $a = \text{old}(\pi(a))$. \hfill \Box

Using this basic meta-language, we can more easily express the relationship between orbits of $\delta(A)$ and orbits of $A$.

Theorem 39. For each $a \in A$, let

$$H^a = \{\text{old}_A(a)\} \cup \{\text{hid}_A^i(a) \mid i \in \text{supp}_A(a)/_\equiv\}$$

$i \equiv j \iff \exists \pi \in G_A(a). \pi(i) = j$

Let $A^o_A$ be a set of canonical representatives of orbits of $A$. A set $A^o_{\delta(A)}$ of canonical representatives of orbits of $\delta(A)$ is obtained as $A^o_{\delta(A)} = \bigcup_{a \in A^o_A} H^a$

Proof. By lemma 38, each element $a$ of $A$ in $\delta(A)$ is represented by an element of $A^o$ as defined above. What we have to show is that the orbits of all elements in $H^a$ are disjoint. First, observe that $i \in \text{supp}_A(a) \implies \text{old}_A(a) \notin \text{orb}_{\delta(A)}(\text{hid}_A^i(a))$ because of the different cardinality of the supports: by theorem 34, $\text{supp}_{\delta(A)}(h^i(a)) = \text{supp}_A(a) \setminus i = \text{supp}_{\delta(A)}(a) \setminus i$. Hence the orbit of the old element is disjoint from the orbit of any hidden element. Now we show that $\text{orb}_{\delta(A)}(h_i(a)) = \text{orb}_{\delta(A)}(h_j(a))$ if and only if
∃π ∈ G_A(a). π(i) = j. For each i ∈ supp_A(a) we have

\begin{align*}
\text{orb}(h_i(a))
= & \{ \pi_A^{-1}(h_i(a)) \mid \pi \in Autf\} \\
= & \{ \pi_A(h_i(a)) \mid \pi \in Autf \land \pi(0) = 0 \} \\
= & \{ (\pi \circ \pi')(a) \mid \pi, \pi' \in Autf \land \pi(0) = 0 \land \pi'(i) = 0 \land \forall j \in \text{supp}(a) \setminus i. \pi'(j) = j + 1 \} \\
= & \{ \pi_A(a) \mid \pi(i) = 0 \}
\end{align*}

Finally, suppose that \( \text{orb}_{g(A)}(h_i(a)) = \text{orb}_{g(A)}(h_j(a)) \). For each \( \pi \) such that \( \pi(j) = 0 \), there exists \( \pi' \) such that \( \pi'(i) = 0 \) and \( \pi_A(a) = \pi'_A(a) \). Then \( a = (\pi^{-1} \circ \pi')_A(a) \) if and only if \( \pi^{-1} \circ \pi' \in G_A(a) \). Observing that \( \pi^{-1} \circ \pi'(j) = i \) we conclude our proof. \( \Box \)

For each orbit in \( A \), represented by \( a_o^A \), there is a corresponding orbit in \( \delta(A) \) without any hidden name (we call elements of these orbits the old elements), plus as many orbits in \( \delta(A) \) as the possible abstractions of names in \( \text{supp}_A(a_o^A) \), modulo its symmetry: there are as many ways to hide a name in \( a_o^A \) as names in its support, up-to an equivalence relation saying that there is no difference in abstracting two names, when they are swapped by some permutations in \( G_A(a_o^A) \) (we call these the hidden elements). The result on orbits also asserts a fundamental property: hidden elements can never be turned into old elements employing the permutation action of \( \delta(A) \), hence they actually are new elements in the resulting algebra. The following example illustrates the need for an existential quantification over \( G_A(a) \).

**Example 40.** Applying the result of theorem 39 to symmetries obtained by round shifts may look counterintuitive. Consider the set of \( \pi \)-calculus agents with names in \( \omega \), up to structural equivalence, seen as a permutation algebra \( P_i = \langle A_{P_i}, \{ \pi_{P_i} \} \rangle \), and agent \( P(1, 2, 3) = \bar{1}2 + 2\bar{3} + \bar{3}1 \). Its symmetry is \( \{ \text{id}, \sigma, \sigma^2 \} \), generated by the round shift \( \sigma(1) = 2, \sigma(2) = 3, \sigma(3) = 1 \). The three agents \( P_1 = (\nu_1)P(1, 2, 3) \), \( P_2 = (\nu_2)P(1, 2, 3) \) and \( P_3 = (\nu_3)P(1, 2, 3) \) belong to the same orbit due to structural equivalence. However, the symmetry of \( P_1 \) (and consequently, of \( P_2 \) and \( P_3 \) which are on the same orbit) is just \( \{ \text{id} \} \): the support of \( P_1 \) is \( \{ 2, 3 \} \), hence the only possible candidate permutation besides the identity is the swap \( \rho(2) = 3, \rho(3) = 2 \), but \( \rho_{P_1}(P_1) = (\nu_1)(\bar{1}3 + 32 + 21) \) which is not structurally equivalent to \( P_1 \) itself, whereas one might have expected \( \rho \in G_{P_i}(P_1) \).
3.3 The Left Adjoint to $\delta$

Being defined by a theory morphism, the name abstraction functor $\delta$ has a left adjoint, which we denote with $F$.

Generally speaking, just like $\delta$, which is a right adjoint, is a forgetful functor, left adjoints over categories of algebras are usually a form of free constructions. Actually, in $\mathcal{F}(\mathcal{A})$, we find a name freely added to the support of each element of $\mathcal{A}$, and as many orbits as in $\mathcal{A}$. The symmetry of each element leaves the freely added name distinct from any other. The counit plays exactly the role to “reveal” a bound name, which is a non-trivial operation due to corollary 29.

To characterise $\mathcal{F}(\mathcal{A})$, we quotient the free algebra over $\mathcal{A}$ using the inference rules of definition 14. The permutation signature is only made up of unary, composable operators, so the presentation of a permutation algebra is in the simple form $\pi(a) = \rho(b)$, for $a, b \in A$, and $\pi, \rho \in Autf$. In the following definition, we represent the free algebra over $A$ simply as the product $A \times Autf$, i.e. we write $\langle a, \pi \rangle$ instead of the usual notation $\pi(a)$ for terms in $T_{Autf}(A)$, to avoid confusion with the notation $\pi_A(a)$. Terms in $T_{Autf}(A)$ are considered quotiented with the axioms of permutation algebras.

**Definition 41** (concretion). The functor $\mathcal{F} : \text{Alg}_\pi \rightarrow \text{Alg}_\pi$ which is left adjoint to $\delta$ is defined on objects as

$$\mathcal{F}(\langle A, \{\pi_A\} \rangle) = \langle T_{Autf}(A)/\equiv, \pi_{\mathcal{F}(A)} \rangle$$

with $\pi_{\mathcal{F}(A)}(\langle a, \rho \rangle) = \langle a, \pi \circ \rho \rangle$, and on arrows as

$$\mathcal{F}(f)(\langle [a, \rho] \rangle) = \langle [f(a), \rho] \rangle$$

The equivalence relation $\equiv$ is the equivalence relation generated by the union of the axioms obtained by the following rule:

$$\pi_A(a) = \rho_A(b) \quad \frac{\langle a, \pi^+ \rangle \equiv \langle b, \rho^+ \rangle}{\langle a, \pi^+ \rangle \equiv \langle b, \rho^+ \rangle}$$

and the axioms of permutation algebras.

We now provide a definition of the unit and the counit of the adjunction. It suffices to exhibit, for each $\mathcal{A}$ an universal arrow $\eta_A : \mathcal{A} \rightarrow \delta(\mathcal{F}(\mathcal{A}))$ such that, for each $f : \mathcal{A} \rightarrow \delta(B)$, there exists a unique $f^#$ making the diagram in Fig. 3.1 commute. From now on, we omit the brackets around canonical representatives of terms, thus we assume that all pairs $\langle a, \rho \rangle$ stand for their canonical representatives in $\mathcal{F}(\mathcal{A})$. 


Theorem 42. An universal arrow $\eta_A : A \to \delta(\mathcal{F}(A))$ is given by

$$\eta_A(a) = \langle a, id \rangle$$

For each $f : A \to \delta(B)$, we have $f^\#: \mathcal{F}(A) \to B$, with $f^\#(\langle a, \rho \rangle) = \rho_B(f(a))$.

Proof. First, we have to prove that $\eta_A$ is an arrow in $\text{FSA}l_g^\pi$, i.e. that $\eta_A(\pi_A(a)) = \pi_{\delta(\mathcal{F}(A))}(\eta_A(a))$. We have $\eta_A(\pi_A(a)) = \langle \pi_A(a), id \rangle = \pi_{\mathcal{F}(A)}^1(\langle a, id \rangle) = \pi_{\delta(\mathcal{F}(A))}(\eta_A(a))$. Now, we prove that $f^\#$ is a permutation algebra morphism, too:

$$f^\#(\pi_{\mathcal{F}(A)}(\langle a, \rho \rangle)) = f^\#(\langle a, \pi \circ \rho \rangle) = (\pi \circ \rho)_B(f(a)) = \pi_B(\rho_B(f(a))) = \pi_B(\langle f^\#(a), \rho \rangle).$$

Finally, we show that the diagram commutes: $(\delta(f^\#) \circ \eta_A)(a) = \delta(f^\#)(\langle a, id \rangle) = f^\#(\langle a, id \rangle) = f(a)$. Uniqueness comes from commutativity, since all elements of the carrier of $\mathcal{F}(A)$, by theorem 45, are in the orbit of $\langle a, id \rangle$ for some $a$, and the value of $f^\#$ on $\langle a, id \rangle$ is uniquely determined by commutativity of the diagram (as shown by the above proof). □

The usual definition of adjunction relies on an isomorphism of homsets, which in our case obtains, for each arrow $f : A \to \delta(B)$, simply the arrow $f^\# : \mathcal{F}(A) \to B$.

The universal arrow $\eta_A$, defined for each object $A$ of $\text{Alg}^\pi$, can be seen as a natural transformation $\eta : Id \to \delta \circ \mathcal{F}$, which is the unit of the adjunction.

The following theorem shows that each element of $\mathcal{F}(A)$ has an additional name, which is added “syntactically” or “freely”, i.e. is not obtained by properties of $A$. In the rest of the section, we assume that all permutation algebras we employ are finitely supported (the fact that $\mathcal{F}$ preserves the finite support property is indeed shown by theorem 43 below).

Theorem 43. The support of an element of $\mathcal{F}(A)$ is given as

$$\text{supp}_{\mathcal{F}(A)}(\langle a, \rho \rangle) = \{\rho(0)\} \cup \{\rho(i + 1) \mid i \in \text{supp}_A(a)\}$$
CHAPTER 3. BEHAVIOURAL FUNCTORS FOR PERMUTATION ALGEBRAS

Proof. First, we show the special case of \( \langle a, id \rangle \). Let \( S_a = \{0\} \cup \{i+1 \mid i \in \text{supp}_A(a)\} \). We have to prove that \( \text{supp}_{\mathcal{F}(A)}(a) = S_a \). We now choose a permutation in \( \text{fix}(S_a) \), which by necessity fixes 0, hence without loss of generality we can assume it is \( \pi^+1 \) for some \( \pi \).

By definition of \( \mathcal{F} \) as a quotient, we have \( \pi^+1 \in \text{fix}(\text{supp}_A(a)) \), hence \( \pi A(a) = a \). From the definition of \( \mathcal{F} \) as a quotient, we have \( \pi^+1 F(\mathcal{F}(A))(\langle a, id \rangle) = \langle a, \pi^+1 \rangle = \langle \pi A(a), id \rangle = \langle a, id \rangle \). Thus, \( \pi^+1 \in \text{fix}(S_a) \Rightarrow \pi^+1 \in \mathcal{G}_{\mathcal{F}(A)}(a) \). We also show that \( S_a \) is the minimum supporting set, by inspection on two cases. If \( \pi \) does not fix 0, \( \pi(\langle a, id \rangle) = \langle a, \pi \rangle \neq \langle a, id \rangle \). We now employ again a permutation fixing 0, so it is in the form \( \pi^+1 \). If \( \pi^+1 \) does not fix \( S_a \setminus 0 \), then it does not support \( a \) in \( \mathcal{A} \), by minimality of support, hence \( \pi A(a) \neq a \).

We have proved our thesis for elements of the form \( \langle a, id \rangle \). Now we have to generalise it to arbitrary elements. We have \( \langle a, \rho \rangle = \rho(\langle a, id \rangle) \), thus the result is generalised.

The symmetry of an element of \( \mathcal{F}(\mathcal{A}) \) is given by the symmetry of \( a \) in \( \mathcal{A} \), translated using the right shift theory morphism and the permutation \( \rho \).

**Theorem 44.** The symmetry of an element \( \langle a, \rho \rangle \) of \( \mathcal{F}(\mathcal{A}) \) is obtained as

\[
\mathcal{G}_{\mathcal{F}(\mathcal{A})}(\langle a, \rho \rangle) = \{\rho \circ \pi^+1 \circ \rho^{-1} \mid \pi \in \mathcal{G}_A(a)\}
\]

Proof. Observe that \( \pi \in \mathcal{G}_A(a) \Rightarrow \pi A(a) = id_A(a) \), then by translation of the axioms we have \( \pi^+1 A(a) = \langle a, \pi^+1 \rangle = \langle a, id \rangle \) in \( \mathcal{F}(\mathcal{A}) \), in other words \( \pi^+1 \in \mathcal{G}_{\mathcal{F}(\mathcal{A})}(\langle a, id \rangle) \). Now we have

\[
(\rho \circ \pi^+1 \circ \rho^{-1})_{\mathcal{F}(\mathcal{A})}(\langle a, \rho \rangle) \\
= (\rho \circ \pi^+1 \circ \rho^{-1})_{\mathcal{F}(\mathcal{A})}(\rho_{\mathcal{A}}(\langle a, id \rangle)) \\
= \rho_{\mathcal{F}(\mathcal{A})}(\pi^+1_{\mathcal{F}(\mathcal{A})}(\langle a, id \rangle)) \\
= \rho_{\mathcal{F}(\mathcal{A})}(\langle a, id \rangle) \\
= \langle a, \rho \rangle
\]

which concludes the proof.

Notice that no permutation in the symmetry can swap (shifted) names of \( a \) and 0, hence 0 is distinguished from names already in \( a \). We finally analyse the set of orbits of \( \mathcal{F}(\mathcal{A}) \), which is isomorphic to that of \( \mathcal{A} \).

**Theorem 45.** Given a finitely supported permutation algebra \( \mathcal{A} = \langle A, \{\pi_A\} \rangle \), a set of canonical representatives of orbits of \( \mathcal{F}(\mathcal{A}) \) is given by \( \mathcal{F}(\mathcal{A})^o = \{\langle a^o, id \rangle \mid a^o \in A^o_A\} \).
Proof. Consider \( a \in A \), and a permutation \( \pi \) such that \( a = \pi_A(a^\circ) \). We have

\[
\langle a, \rho \rangle = (\text{by definition of permutation action in } F) \\
\rho_{F(A)}(\langle a, \text{id} \rangle) = (\text{by the quotient of definition 41}) \\
\rho_{F(A)}(\langle a^\circ, \pi^{+1} \rangle) = (\rho \circ \pi^{+1})_{F(A)}(\langle a^\circ, \text{id} \rangle)
\]

thus we have found a permutation \( \sigma = \rho \circ \pi^{+1} \) such that \( \langle a, \rho \rangle = \sigma_{F(A)}(\langle a^\circ, \text{id} \rangle) \). \qed

For each \( a \in A \) we can recover, as we did for abstraction (employing old), an element having the same properties of \( a \) in \( A \), plus the addition of a new name.

**Lemma 46.** For each element \( a \), and each name \( i \notin \text{supp}_A(a) \) there exists an element of \( F(A) \) whose support is \( \text{supp}_A(a) \cup \{i\} \), with \( G_{F(A)}(a) = G_A(a) \cap \text{fix}(i) \).

**Proof.** Consider an element \( a \in A \), and for readability, let \( s = \text{supp}_A(a) \), and \( s^{+1} = \{i + 1 \mid i \in s\} \). In a pair \( \langle a, \text{id} \rangle \), \( s \) is shifted to \( s^{+1} \), and 0 is added as a free name. Then, no permutation \( \rho \) can fuse 0 with any element of the \( s \). In particular, for each name \( i \) in \( \omega \setminus (s \cup \{0\}) \), we have a permutation \( \sigma^{(i)} \) that sends back all the names in \( s^{+1} \) to \( s \), and sends 0 to \( i \). The support of the obtained pair \( \langle a, \sigma^{(i)} \rangle \) is simply \( \text{supp}_A(a) \cup \{i\} \) by theorem 43. \qed

### 3.3.1 The Counit as a concretion operation

The counit is given by \( \epsilon : F \circ \delta \to Id \), such that \( \epsilon_A = \text{id}^\#_{\delta(A)} \). By expanding the definition, we obtain the following:

**Definition 47 (counit).** The counit \( \epsilon_A : F(\delta(A)) \to A \) of the adjunction between \( \delta \) and \( F \) is defined, for each permutation algebra \( A \), as

\[
\epsilon_A(\langle a, \rho \rangle) = \rho_A(a)
\]

The role of the counit is to reveal a hidden name. To explain how it is so, we can start by looking at lemma 29. Because of it, no morphism can send an element \( a \) of \( \delta(A) \) having an hidden name into itself in \( A \), having no hidden names: that would add one
name to the support of the destination. Hence, the only way to reveal a fresh name is to employ the product algebra $\omega \times A$, and in particular pairs of the form $\langle n, a \rangle$ where name $n$ is fresh in $a$. Now a morphism $f : \delta(A) \rightarrow A$ can reveal names as follows: if $a$ has no bound name, then we pose $f(\langle n, a \rangle) = a$. If $a$ has a bound name, that is, $0 \in supp_A(a)$, then we have $f(n, a) = \sigma^{(0,n)}_A(a)$, where $\sigma^{(n,m)}$ is the permutation swapping $n$ and $m$. The counit does exactly this, by virtue of the following lemma (a simple consequence of lemma 46), which gives a more intuitive description of the permutation algebra $F(A)$.

**Lemma 48.** Given a finitely supported permutation algebra $A = \langle A, \{\pi_A\} \rangle$, the algebra $F(A)$ is isomorphic to

$$\langle \hat{A}, \{\pi_\times\} \rangle$$

where $\pi_\times$ is the standard permutation action over the product $\omega \times A$, restricted to its subalgebra

$$\hat{A} = \{ \langle n, a \rangle \mid a \in A \land n \notin supp_A(a) \}$$

The action on the counit, composed with the above isomorphism, is exactly the operation that we described above that reveals hidden names.

### 3.4 Comparison with other definitions

Here we provide the due comparison between the notions we have just obtained and the nominal set of abstractions and the freshness relation of nominal sets. The notions turn out to be isomorphic (hence, we get a proof of another folklore result, the adjunction between the two functors in nominal syntax). Nevertheless, the abstraction functor we provided is worth being defined for an interesting side result: in §4.1 we describe the syntax of the $\lambda$-calculus as an initial algebra for a functor employing $\delta$, thus filling the gap between nominal abstract syntax and De Bruijn indexes.

Abstraction $[i]a$ for an element $a$ of a nominal set $A$ and a name $i$ (see [GP02]) is defined as the equivalence class obtained by swapping $i$ with a name $j$ in the pair $\langle i, a \rangle$, for all possible names $j$ not in the support of $a$. This is quite the idea of representing the axioms of $\alpha$-conversion, while the idea of shifting names, and calling the bound name 0, is typical of the De Bruijn indexes approach [dB72].

Without the need to use a mathematical formalism, it should be self-evident at this point that, by theorem 30, objects obtained from the two constructions are isomorphic.
As an immediate and important consequence, we have that \( \delta \) is an accessible functor: as shown in [Men03], the abstraction functor of Gabbay and Pitts is both a left and a right adjoint. This implies that it preserves all limits and colimits, and in particular \( \lambda \)-filtered colimits. This can also be stated by equivalence with the corresponding functor in the Schanuel topos, however we state it as a theorem for its relevance in the rest of the work.

**Theorem 49.** The name abstraction functor \( \delta \) is accessible.

On the other hand, the *freshness relation* \( i \# a \) in nominal syntax is defined as the set of elements \( a \) paired with names \( i \) which are *fresh* for \( a \), i.e. not in its support. This is a nominal set when equipped with the ordinary action of the permutation on the product. By theorem 48 the nominal set of freshness is isomorphic to the object \( F(A) \). Hence, we may conclude this section by saying the adjunction associated to the definition of the right shift theory morphism provides a proof of the adjunction between abstraction and concretion in the setting of nominal sets, and finds a place in the diagram for the concretion operation.

### 3.5 Product, Coproduct and Power Set

Abstraction and concretion are peculiar features of categories that deal with names and name generation, such as permutation algebras or the Schanuel topos. Now we present other endofunctors that are already present in the category \( \text{Set} \), and in particular product, coproduct and power set. The first two are just lifted from \( \text{Set} \), while the latter, in the finitely supported case, has an interesting behaviour that gives rise to a sort of name binding in turn.

#### 3.5.1 Product and Coproduct

In a presheaf category \( \text{Set}^C \) over a small category \( C \), pointwise (co)limits are (co)limits. Consider a diagram \( D \) in \( \text{Set}^C \). For each object \( o \) of \( C \) we have a “section” \( D_o \) of the diagram, which is a diagram in \( \text{Set} \). If, for each \( o \), a limit \( L_o \) of \( D_o \) exists in \( \text{Set} \), then the presheaf \( F(o) = L(o) \) is a limit of \( D \) in \( \text{Set}^C \). This argument applies to product and coproduct. Moreover, if \( C \) has only one object, then all this amounts to say that limits and colimits in \( \text{Set} \) also are limits and colimits in \( \text{Set}^C \). For product
and coproduct in $\textbf{Set}^\text{Aut}$, we thus have the following definitions, where we assume the algebras $\mathcal{A} = \langle A, \{\pi_A\} \rangle$ and $\mathcal{B} = \langle B, \{\pi_B\} \rangle$.

**Definition 50** (product). The product of $\mathcal{A}$ and $\mathcal{B}$ is obtained as

$$\langle A \times B, \{\langle \pi_A; \pi_B \rangle \} \rangle$$

where the action of a permutation $\pi$ is the pairing of $\pi_A$ and $\pi_B$.

**Definition 51** (coproduct). The coproduct of $\mathcal{A}$ and $\mathcal{B}$ is defined as

$$\langle A + B, \{[\pi_A; \pi_B]\} \rangle$$

where the action of a permutation $\pi$ is the copairing of $\pi_A$ and $\pi_B$.

The pairing and copairing of two permutation algebra morphisms (giving rise to the action of the functors on arrows) are just the corresponding set-theoretical notions. Algebraic properties of the coproduct (i.e. symmetry and orbits of elements) are trivially defined using the copairing of the symmetry, and the copairing of the set of orbits. For the product, instead, we have an interesting property: the symmetry of a pair $\langle a,b \rangle$ is obtained as the intersection of the symmetries of $a$ and $b$.

**Theorem 52.** Given two permutation algebras $\mathcal{A} = \langle A, \{\pi_A\} \rangle$ and $\mathcal{B} = \langle B, \{\pi_B\} \rangle$, the symmetry of an element $\langle a,b \rangle$ in $A \times B$ is given by

$$\mathcal{G}_{A \times B}(\langle a,b \rangle) = \mathcal{G}_A \cap \mathcal{G}_B$$

**Proof.** Just observe that $\pi_{A \times B}(\langle a,b \rangle) = \langle a,b \rangle \iff \pi_A(a) = a \land \pi_B(b) = b$. □

A simple corollary is the following.

**Corollary 53.** It holds that $\text{supp}_{A \times B} = \text{supp}_A(a) \cup \text{supp}_B(b)$.

### 3.5.2 The Power Set

Here we will give the definition of the covariant and contravariant power set over permutation algebras. We start with the non finitely-supported case, where the definitions just lift from $\textbf{Set}$. First, we deal with the covariant functor.
Definition 54 (covariant power set). The covariant power set of a permutation algebra \( \langle A, \{\pi_A\} \rangle \) is defined on objects as \( \langle \mathcal{P}(A), \{\pi_{\mathcal{P}(A)}\} \rangle \) where the permutation action is defined as \( \pi_{\mathcal{P}(A)}(p) = \{\pi_A(a) \mid a \in p\} \). The action of the functor on arrows is \( \mathcal{P}(f)(p) = \{f(a) \mid a \in p\} \).

The countable power set functor \( \mathcal{P}_\omega \) can also be defined in the obvious way. However, it does not restrict to an endofunctor in \( \text{FSAlg}^\pi \). For example, the set of odd numbers is not finitely supported as an element of \( \mathcal{P}_\omega(\omega) \). This motivates the definition of a finitely supported power set, which we denote with \( \mathcal{P}_{fs} \).

Definition 55 (finitely supported power set). The finitely supported power set

\[
\mathcal{P}_\omega : \text{FSAlg}^\pi \to \text{FSAlg}^\pi
\]

is the restriction of the countable power set in \( \text{Alg}^\pi \) to finitely supported subsets of the carrier set: elements of \( \mathcal{P}_\omega(A) \) are all the elements of \( \mathcal{P}(A) \) having the finite support property.

Notice that being countable is not connected to having finite support, however we define the power set in this way because we only need the countable version in the rest of this thesis.

The restriction to finite support has an important effect. Consider a permutation algebra \( \langle A, \{\pi_A\} \rangle \). A set \( p \subseteq A \) is finitely supported when, for a cofinite set \( \bar{S} \subseteq \omega \), that is, such that \( S = \omega \setminus \bar{S} \) is finite, for all \( \rho \in \text{fix}(S) \), we have \( \rho_{\pi_A}(p) = p \). This happens not only when all the elements of \( p \) are supported by \( S \), but also when each element \( a \in p \) is supported by a set \( \text{supp}_A(a) \) larger than \( S \), provided that \( p \) is closed with respect to all permutations sending names in \( \text{supp}_A(a) \setminus \{i\} \) into any element of \( \omega \setminus S \).

Example 56. Let \( \langle A, \{\pi_A\} \rangle \) be a permutation algebra, and \( \sigma^{(i,j)} \) a permutation swapping \( i \) with \( j \), \( a \in A \), \( i \in \text{supp}_A(a) \). The set \( r = \{\sigma_A^{(i,j)}(a) \mid j \in (\omega \setminus \text{supp}_A(a)) \cup \{i\}\} \) is finitely supported by \( \text{supp}_A(a) \setminus \{i\} \) for \( \rho \in \text{fix}(\text{supp}_A(a) \setminus \{i\}) \), we have \( \rho_A(a) \mid a \in r = r \), i.e. the name \( i \) “disappears” from the support of \( r \).

In such a case, \( p \) itself is infinite but, we may say, “well behaved” since it is obtained from a finite set, whose elements are permuted in an infinite number of ways. To give a formal account of this, we show the following theorem.
Theorem 57. Let \( p \in \mathcal{P}(A) \) be finitely supported with support \( S \). Then, for each \( a \in p \), either \( \text{supp}_A(a) \subseteq S \) or for all \( \rho \) such that \( \rho(\text{supp}_A(a)) \setminus S \subseteq \omega \setminus S \), and \( \rho(S) = S \), we have \( \rho_A(a) \in p \).

Proof. We have \( \text{fix}(S) \subseteq G_{\mathcal{P}(A)}(p) \), and \( \rho(\text{supp}_A(a)) \setminus S \subseteq \omega \setminus S \land \rho(S) = S \) implies \( \rho \in \text{fix}(S) \). Hence, \( \rho_{\mathcal{P}(A)}(p) = p \), thus for all \( a \in p \), \( \rho(a) \in p \), from which the thesis.

What the above theorem says is that each element of \( p \) having support greater than that of \( p \) is necessarily replicated infinitely many times, in such a way that these additional names disappear from the support of the whole \( p \).

For completeness, we introduce also the contravariant functor. To show that the contravariant power set is indeed a functor in \( \text{FSAlg}^\pi \), we need the following lemma.

Lemma 58. Given a morphism \( f : A \to B \) in \( \text{FSAlg}^\pi \), its set-theoretical inverse \( f^{-1} : \mathcal{P}(B) \to \mathcal{P}(A) \), defined as

\[
f^{-1}(p) = \{ a \in A \mid f(a) \in p \}
\]

is an arrow in \( \text{FSAlg}^\pi \).

Proof. We have to show that \( f^{-1} \) respect the permutation action of the power set. We have:

\[
f^{-1}(\pi_{\mathcal{P}(p)}(p)) = \{ a \in A \mid f(a) \in \pi_{\mathcal{P}(p)}(p) \}
= \{ a \in A \mid f(a) = \pi_B(b) \land b \in p \}
= \{ \pi_A(a') \mid a' \in A \land f(\pi_A(a')) = \pi_B(b) \land b \in p \}
= \{ \pi_A(a') \mid a' \in A \land \pi_B(f(a')) = \pi_B(b) \land b \in p \}
= \{ \pi_A(a') \mid a' \in A \land f(a') \in p \}
= \pi_{\mathcal{P}(f^{-1}(p))}
\]

The definition of the functor is as follows.

Definition 59 (contravariant power set). The finitely supported contravariant power set acts on each object \( A \) as \( \mathcal{P}_\omega(A) \), and on each arrow \( f \) as the set-theoretical inverse \( f^{-1} \).
Chapter 4

Examples

4.1 Example: De Bruijn indexes

To illustrate how De Bruijn indexes are obtained using $\delta$, we show an encoding of the syntax of the $\lambda$-calculus. We assume finite products and coproducts to be defined in $\text{FSAlg}^\pi$ (these functors are trivially lifted from $\text{Set}$, defining the permutation operation pointwise), and, given a finite coproduct $X_0 + X_1 + \ldots + X_n$ we denote its elements with $\{\langle i, x \rangle \mid i \in \{0, \ldots, n\} \land x \in X_i\}$.

Consider the set of $\lambda$-calculus terms, defined by the syntax

$$L ::= \lambda x.L \mid LL \mid x$$

for $x \in \omega$. Instead of introducing the notion of $\alpha$-equivalence for terms, we can define the syntax as the term algebra $\Lambda$ of a functor $T$, i.e. as an arrow in $\text{FSAlg}^\pi$ of type $T(X) \to X$. The functor we chose is

$$T(X) = \delta(X) + X \times X + \omega$$

where $\omega$ is seen as a permutation algebra with $\pi_\omega(i) = \pi(i)$. Notice that, being $T$ a functor in $\text{FSAlg}^\pi$, an action of the permutation $\pi_\Lambda$ on $\Lambda$ is defined by initiality using $\pi_\omega$ as the base case, thus introducing support, symmetry and orbits of elements. An interpretation $f$ of $\lambda$-terms as elements of $\Lambda$ is given in Fig. 4.1.

The most important case is the usage of $\text{hid}_\Lambda$ to define $f(\lambda x.l)$: it shifts all names of $l$ by one, and assigns the name 0 to the bound name. There is more: all the $\alpha$-equivalent terms of the form $\lambda y.l[\gamma/x]$ are translated into the same element $t$ of $\Lambda$, where name 0
is not “visible” in $supp_{\Lambda}(t)$, and permutations can not exchange it with any other. This corresponds to a notion of “capture-avoiding permutation” for $\Lambda$. In fact, by theorem 34, we have $supp_{\Lambda}(t) = supp_{\lambda}(f(t)) \setminus \{x\}$.

### 4.2 Example: the $\pi$-calculus

In this section, we show how $\delta$ can be used to express the early semantics of the $\pi$-calculus as a coalgebra in $\text{FSAlg}^{\pi}$, hence in nominal sets. We closely follow [MP05], where the early semantics is given in the form of Structural Operational Semantics rules, implicitly defining a functor for the $\pi$-calculus. Results in [CHM02] ensure a lifting of the semantics from the category $\text{Set}$ to the category $\text{Alg}^{\pi}$ that respect axioms, obtaining a bialgebra where the algebraic operations and axioms are those of the permutation signature.

Our contribution here is to give an explicit form for the functor employed in [MP05]. We also explain how this functor can be more conveniently expressed as $P_{fs}(L \times \delta(-))$. This is indeed possible also in $\text{Set}$. Coalgebras for this functor in $\text{FSAlg}^{\pi}$ are very similar to transition systems in $\text{Set}$. The only two additions that we have is that states have a support (and a symmetry), and transitions may allocate fresh names. This is very similar to the kind of “enriched” state machines that are sometimes used to depict protocols that feature resource allocation, e.g. security protocols.

Notice that using $P_{fs}$ actually addresses a potential question that might arise on the results in [MP05]: the lifting of the semantics is to $\text{Alg}^{\pi}$, not $\text{FSAlg}^{\pi}$. However, the framework of [CHM02] is applied to ensure that a final coalgebra semantics of the early $\pi$-calculus exists as a finitely supported permutation algebra. The reason why this holds is that $\text{FSAlg}^{\pi}$ is a full subcategory of $\text{Alg}^{\pi}$, hence the image of each morphism yields a finitely supported algebra out of any finitely supported algebra.

In the following, we define two permutation algebras of labels, with $x$ and $y$ ranging over $\omega$, having syntactic substitution as the permutation action $\pi_{L'}$: $L'$ with carrier \{tau, in(x, y), out(x, y)\} and $L''$ with carrier \{bout(x)\}. The transition specification $\Delta_{\pi}$

\[
\begin{align*}
f(\lambda x.l) &= \langle 0, hid^{\pi}_{\Lambda}(f(l)) \rangle & \text{if } x \in fn(l) \\
f(\lambda x.l) &= \langle 0, old^{\pi}_{\Lambda}(f(l)) \rangle & \text{if } x \notin fn(l)
\end{align*}
\]

\[
\begin{align*}
f(l_1 l_2) &= \langle 1, (f(l_1), f(l_2)) \rangle \\
f(x) &= \langle 2, x \rangle
\end{align*}
\]
4.2. EXAMPLE: THE $\pi$-CALCULUS

is the same reported in Fig. 2.5.

**Theorem 60.** Let $T(X) = P_{\mathcal{L}}(\mathcal{L}' \times X + \mathcal{L}'' \times \delta(X))$. A transition function in $\text{FSAlg}^\pi$ is a coalgebra for $T$ if and only if it respect $\Delta_\pi$.

**Proof.** For the “only if” part, let $A = (A, \pi_A)$ be a permutation algebra, and $f : A \to T(A)$. We show that $f$ respect the transition specification $\Delta_\pi$. We adopt the notation $\langle 0, x \rangle$ to describe the left component of the coproduct, and $\langle 1, x \rangle$ to describe the right component. The set of outgoing transitions from an element $a \in A$ is thus made up of elements of the form $\langle 0, \langle l, a \rangle \rangle$, where $a \in A$ and $l \in \mathcal{L}'$, and of elements of the form $\langle 1, \langle \text{bout}(x), a \rangle \rangle$, with $a$ element of $\delta(A)$. The proof derives directly from the fact that $f$ is a permutation algebra morphism.

In the first case, the action of a permutation on a transition respect rule 1:

$$\langle 0, \langle l, a \rangle \rangle \in f(a) \quad \Rightarrow \quad \rho_{\mathcal{L}' \times A}(\langle 0, \langle l, a \rangle \rangle) \in f(\rho_A(a))$$

$$\iff \langle 0, \langle \rho(l), \rho(a) \rangle \rangle \in f(\rho_A(a))$$

In the second case the action of a permutation respect rule 2:

$$\langle 1, \langle \text{bout}(x), a \rangle \rangle \in f(a) \quad \Rightarrow \quad \rho_{\mathcal{L}'' \times \delta(A)}(\langle 1, \langle \text{bout}(x), a \rangle \rangle) \in f(\rho_A(a))$$

$$\iff \langle 1, \langle \text{bout}(\rho(x)), \rho^{+1}(a) \rangle \rangle \in f(\rho_A(a))$$

For the “if” part, a sketch may be given as follows. Observe that if a transition function is in $\text{FSAlg}^\pi$, then it is a transition system with a finitely supported set of transitions. This is because finitely supported permutation algebras are a full subcategory of permutation algebras, hence the transition function has to preserve finiteness of the support. Obeying to either meta-rule 1 or 2, and being a labelled transition system, brings in the coproduct of products. Now one observes that the transition function, being a morphism, has to preserve the permutation action, and meta-rules specify exactly the permutation action in the destination state of a transition. Rule 1 specifies that the permutation action is unchanged, hence we obtain the identity functor, while rule 2 specifies that the permutation action must act, in the target, as $\rho^{+1}$ for each permutation which acts as $\rho$ in the source. This is the definition of $\delta$. 

$\square$
An arrow $f : P_i \to T(P_i)$ representing the semantics of the $\pi$-calculus is described by the rules in Fig. 4.2, where we do not use additional labels to discriminate the two components of the coproduct, since the labels in $L'$ and $L''$ are disjoint. As in [MP05], we employ the transition system in $\textbf{Set}$ in premises of the rules. Notice that permutation $\sigma$, applied to $p'$ in rule for bound output in [MP05], corresponds exactly to $\text{hid}_{P_i}^p(p')$, hence the coalgebra provided there is the same as the one we introduced.

By posing $L = L' \cup L''$, and using the embedding $\text{old}$ of $X$ into $\delta(X)$, the functor $T(X) = \mathcal{P}_{fs}(L \times \delta(X))$ can as well be used. This shows that a binding functor $\delta$ over destination states is essentially the only addition to ordinary LTSs which is needed to represent the early semantics of the $\pi$-calculus in a purely coalgebraic way. In $\textbf{FSAlg}^\pi$, bisimulation for the early $\pi$-calculus is the coalgebraic notion, without any side condition on bound names, due to proper handling of fresh names. Assuming that $\text{old}$ is implicitly used on transitions unless a binding is performed, one can define “abstract semantics with binding” by just giving ordinary transition rules over the underlying set of a nominal set, using the operation $\text{hid}^d$ whenever a fresh name is needed in the destination.

The contents of this section might thus be summarised as showing that the category of nominal sets, together with the abstraction functor, allows one to enrich in a natural way ordinary labelled transition systems with operators for dynamic allocation of names.
Part III

History Dependent Automata
Introduction

The theory of history-dependent automata (HD-automata for short) has its roots in the works on the location semantics of the CCS [MPY96] and the causal semantics of Petri nets [MP97], and in the general idea of tracing the history of resources along operational steps in computation. The main achievement in this respect is the idea that resources, abstractly identified with names as it is usual in nominal formalisms, are local to system states. This is different from e.g. the semantics of the π-calculus [MPW92], where public channel names are assumed to be universally known. In history-dependent models, the same identifier can assume different meanings at different times during the history of execution.

On one hand, local names model in a natural way those formalisms where the fact that two elements have different identity is more important than the actual name given to them. An example is given by CCS with localities [BCHK93]. In this calculus, locations are merely placeholders to represent an execution context.

On the other hand, local names are also useful to give the semantics of calculi with global names, because they allow algorithms that manipulate the semantics to be implemented: the possibility to reuse an old name introduces garbage collection of fresh names that are no longer in use. This is a feature that is lacking in ordinary presheaf models for nominal calculi such as $\text{Set}^I$ or permutation algebras.

One of the ideas that make HD-automata appealing is the possibility to use them as an “intermediate” operational model, in which to map various different nominal calculi. An intermediate computational model expressive enough to represent fresh resource generation is especially useful in the context of service-oriented computing, to be able to publish a language-neutral specification of an abstraction of a service behaviour. This point of view has not been abandoned. In [CFPT08], for example, the author, Emilio Tuosto, Marco Pistore and Gian Luigi Ferrari defined a symmetrical service binding mechanism directly on HD-automata, aimed at a form of protocol-level versioning. The aim in doing so is to obtain behavioural interface descriptions that are independent from the specification language in use at each service provider.

In [FS06] and [GMM06], it was shown that named sets, permutation algebras and the Schanuel topos are related by an equivalence of categories. The major implication of such a result, from our point of view, is that it is possible to translate any semantics developed using presheaf categories with global names into HD-automata.
Even though the model used for the $\pi$-calculus in [FMT05a] is coalgebraic, the functor used there was specifically defined, i.e. it was not obtained by composition of basic functors such as the product or power set. In this part of the thesis, we rectify this by defining a language of accessible endofunctors for named sets. Moreover, equivalence between defined functors and the corresponding ones in $\text{FSAlg}^\pi$ allows us to extend previous equivalence results to categories of coalgebras for these functors, and to show existence of a final coalgebra for all the obtained functors.
Chapter 5

Named Sets and HD-Automata

Here we introduce the category \( \text{NSet} \) of named sets, that are sets equipped with a notion of \textit{names} and \textit{symmetry} associated to each element. The most distinguishing feature of named sets, \textit{locality} of names, is best explained in terms of arrows of the category: these are enriched set-theoretical functions, that carry, for each element of the destination, a \textit{renaming} specifying how the meaning of names of the target is obtained from that of the source, that is, the \textit{history} of names along morphisms. This is exploited in coalgebras over named sets, called \textit{history-dependent automata}, to trace the history of local names in transitions. Similarly to [FS06], we use a cleaner notation for named sets with respect to previous works, introducing an intermediate category, called \textit{Symset}, whose objects represent finite sets of names with an associated symmetry, and whose arrows represent renamings associated to morphisms. With respect to previous works, we give up on the explicit representation of names of canonical representatives of orbits as intervals of the form \( \{0,\ldots,n\} \), obtaining a much simpler definition without substantially affecting the theory.

Symmetries makes the definitions more involved, but, as we discussed before, they are necessary. In models with local names, one should not “escape” symmetry by providing explicit handling of names (e.g. lists or sets of names), which is often done when trying to model memory allocation and deallocation in programming languages. Doing so results in too concrete representations, because it makes possible to observe, as separate entities, names that are indistinguishable.
5.1 The Category Symset

Objects of Symset are groups of permutations over finite sets. Arrows represent injective renamings, thus one might expect these to be injective functions. However, we should not distinguish two renamings when they can be made equal by a permutation in the symmetry of its source. Thus, an arrow from $\phi_1$ to $\phi_2$ is an equivalence class of functions, which is obtained as the composition $i \circ \phi_1$, where $i$ is an injection. This identifies all the possible variants of a renaming obtained by composition with permutations in $\Phi_1$.

In this work, for readability, we use the notation for function composition also to compose sets of functions by composing all the possible pairs, considering single functions in such cases as singleton sets, i.e. $i \circ \phi$ stands for $\{i \circ \sigma \mid \sigma \in \phi\}$. Also the notations $\text{dom}(F)$ and $\text{cod}(F)$ are extended from single functions to sets of functions when all of those have the same domain and codomain. In particular this enables us to denote with $\text{dom}(\Phi)$ the underlying set of a group of permutations $\Phi$. Moreover, we denote with $\text{Grp}(S)$ the set of all permutation groups over $S$.

**Definition 61 (Symset).** Objects of the category Symset are groups

$$ \{\Phi \in \text{Grp}(S) \mid S \in \mathcal{P}_{\text{fin}}(\omega)\} $$

An arrow from $\Phi_1$ to $\Phi_2$ is a set of injective functions $i \circ \Phi_1$ such that

$$ i : \text{dom}(\Phi_1) \xrightarrow{\text{inj}} \text{dom}(\Phi_2) $$

$$ \Phi_2 \circ i \subseteq i \circ \Phi_1 $$

The additional condition in the definition of arrows can be given an intuition by observing that it is equivalent to $\Phi_2|_{\text{Im}(i)} \subseteq i \circ \Phi_1 \circ i^{-1}$, i.e. $\Phi_2$ has a subgroup which is isomorphic to $\Phi_1$. This is necessary to represent permutation algebras, due to theorem 28, and makes arrows closed under composition: for $G = i_2 \circ \Phi_2$ and $F = i_1 \circ \Phi_1$, we have $G \circ F = (i_2 \circ i_1) \circ \Phi_1$ (see proof of theorem 63). Notice the restriction of $\Phi_2$ to $\text{Im}(i)$ in the above inclusion: since the arrows of Symset trace the (backward) history of names along morphisms, the names that are not in $\text{Im}(i)$ are those that are discarded, and there is no necessity to put constraints on them.

**Definition 62 (Identity and composition).** The identity is defined as $\text{id}_\Phi = \Phi$ and composition is defined as

$$ G \circ F = \{g \circ f \mid g \in G \land f \in F\} $$
Theorem 63. Symset is a category. The category $\mathbf{I}$ of finite sets and injective functions is isomorphic to a full subcategory of it.

Proof. The identity laws are trivially obtained by closedness properties of permutation groups. Now we prove that composition actually yields an arrow. Let $F_1 : \Phi_1 \to \Phi_2 = i_1 \circ \Phi_1$ and $F_2 : \Phi_2 \to \Phi_3 = i_2 \circ \Phi_2$ be two arrows.

We first show that $\Phi_2 \circ i_1 \circ \Phi_1 = i_1 \circ \Phi_1$. Since $F_1$ is a morphism, we have $\Phi_2 \circ i_1 \subseteq i_1 \circ \Phi_1$. Hence $\Phi_2 \circ i_1 \circ \Phi_1 = i_1 \circ \Phi_1$. The left part of this equation is, by definition, $F_2 \circ F_1$.

On the other hand, $i_1 \circ \Phi_1 \subseteq \Phi_2 \circ i_1 \circ \Phi_1$ because $id \in \Phi_2$. Hence $\Phi_2 \circ i_1 \circ \Phi_1 = i_1 \circ \Phi_1$. From this, it comes that $i_2 \circ \Phi_2 \circ i_1 \circ \Phi_1 = i_2 \circ i_1 \circ \Phi_1$. The left part of this equation is, by definition, $F_2 \circ F_1$.

Now we have to show the condition $\Phi_3 \circ (i_2 \circ i_1) \subseteq (i_2 \circ i_1) \circ \Phi_1$. We have $\Phi_3 \circ i_2 \subseteq i_2 \circ \Phi_2$, thus $\Phi_3 \circ i_2 \circ i_1 \subseteq i_2 \circ (\Phi_2 \circ i_1) \subseteq i_2 \circ (i_1 \circ \Phi_1)$.

Finally, we also show that $\mathbf{I}$ is isomorphic to a full subcategory of Symset, take the full subcategory of Symset whose objects are of the form $\{id_S\}$ for $S \in \mathcal{P}_{fin}(\omega)$. Indeed $\mathbf{I}$ is isomorphic to it: for each object $S$ of $\mathbf{I}$ just take the object $\{id_S\}$. For each arrow $i : S \to T$ of $\mathbf{I}$ take the set $\{i\}$, which trivially satisfies the required conditions to be an arrow in in Symset. Notice that all the arrows between $\{id_S\}$ and $\{id_T\}$ in Symset are in this form. □

5.2 Named Sets and History Dependent Automata

Employing Symset, named sets are easily defined.

Definition 64 (named set). A named set is a pair

$$N = (Q_N, S_N)$$

where $Q_N$ is a set, and $S_N : Q_N \to |\text{Symset}|$ is a function associating an object of Symset to each element of $Q_N$.

From now on, we call $||q||_N$ the set $\text{dom}(S_N(q))$ of names of $q$. The definition of named function is given as follows.
Definition 65 (named function). A named function from $N$ to $M$ is a pair

$$ F = \langle h_F, \Sigma_F \rangle $$

where

$$ h_F : Q_N \to Q_M $$

and $\Sigma$ is a function, dependently typed on $Q_N$, that returns for each $q \in Q_N$ a morphism of Symset such that

$$ \forall q \in Q_N. \Sigma(q) : S_M(h_F(q)) \to S_N(q) $$

A named function $F$ has two components, $h_F$ and $\Sigma_F$. The former gives an ordinary function over $Q_N$. The latter, for each $q \in Q_N$, gives an arrow of Symset, denoting a backward mapping (the history of names) relating names of $h_F(q)$ to names of $q$. The identity named function is defined as $id_N = \langle id_{Q_N}, \lambda q. id_{S_N(q)} \rangle$. Given two arrows $F : N \to M$ and $G : M \to O$, their composition is defined as $G \circ F = \langle h_G \circ h_F, \lambda q. \Sigma_F(q) \circ \Sigma_G(h_F(q)) \rangle$, as shown in Fig. 5.1.

Theorem 66. Named sets and named functions form a category, called NSet.

The proof is trivial, once shown that Symset is a category in turn (see theorem 63). A mathematical description of stateful systems with name allocation is given by History-Dependent automata (HD-automata for short).

Definition 67 (HD-automaton). A history-dependent automaton is a coalgebra in NSet, i.e. a named function

$$ F : N \to T(N) $$

for $N$ named set and $T$ some endofunctor over NSet.
5.3 Equivalence with FSAlg

The main intuition behind equivalence of FSAlg and NSet is as follows: in FSAlg, names have a global, immutable identity. This property is what makes each element of an orbit distinguished from all the others. To discard globality of names, just a very simple operation that is needed, that is, to quotient orbits by saying that all elements of an orbit have the same meaning. This way, elements that differ only by an injective relabelling are no longer distinct. However, to avoid quotienting morphisms that map orbits in the same way, but differ in which element of the destination orbit is associated to each element of the source orbit, the history of names has to be associated to each morphism, as a form of “tag” to differentiate it.

Example 68. Consider the permutation algebra \( \omega \) (equipped with the natural permutation action) the algebra \( \omega \times \omega \), and the two projections \( \pi_1 \) and \( \pi_2 \). The product \( \omega \times \omega \) only has two orbits, that can be canonically represented by the elements \( \langle 0, 0 \rangle \) and \( \langle 0, 1 \rangle \), corresponding to say that the two involved names either are equal or differ. The algebra \( \omega \) only has a trivial orbit represented by \( 0 \). The arrows \( \pi_1 \) and \( \pi_2 \) map orbits to orbits in the same way, that is, they map each orbit into the only possible value \( 0 \). By augmenting these with the history of names, we are able to distinguish them: the history of names of \( \pi_1 \) when applied to \( \langle 0, 1 \rangle \) maps \( 0 \) to \( 0 \), while that of \( \pi_2 \) maps \( 0 \) to \( 1 \). The history of names of both morphisms, when applied to \( \langle 0, 0 \rangle \), maps \( 0 \) to \( 0 \) in both cases, as the only possible choice.

We will get back to this example in §6.4 after having defined the categorical product of named sets. We now recall the equivalence result proved as proposition 29 in [GMM06].

Theorem 69. The categories NSet and FSAlg are equivalent.

The functor \( F : \text{FSAlg} \to \text{NSet} \) shows how named sets and named functions can be built from permutation algebras and their morphisms.

Definition 70 (from FSAlg to NSet). \( F \) sends each permutation algebra \( \langle A, \{ \pi_A \} \rangle \) into the named set

\[
F(\langle A, \{ \pi_A \} \rangle) = \langle A^o, S_N \rangle
\]

where the symmetry \( S_N \) is defined as

\[
S_N(a^o) = g_A(a^o)_{\supp A(a^o)}
\]
Each arrow \( f : \mathcal{A} \to \mathcal{B} \) is sent by \( F \) into the named function

\[
\langle h_f, \Sigma_f \rangle
\]

where the underlying function and the name mappings are defined as

\[
h_f(a^o) = f(a^o) \\
\Sigma_f(a^o) = \rho \circ S_N(f(a^o))
\]

and \( \rho \) is such that \( \rho_B(f(a^o)) = f(a^o) \).

Each orbit of a permutation algebra “collapses” into its canonical representative in the action of \( F \) on objects, thus discarding the global meaning of names: all elements that only differ for an injective renaming are identified. The history of names in named sets is necessary quite because of such a quotient: since a choice for \( a^o \) and \( f(a^o) \) has been made, we can no longer distinguish between two different functions that map orbits in the same way, but select different element of each orbit.

**Example 71.** The permutation algebra of natural numbers \( \omega \) has only one orbit, having just one name in the support of each element, and trivial symmetry. The corresponding named set \( \langle Q_\omega, S_\omega \rangle \) is thus defined by \( Q_\omega = \{0\} \), \( S_\omega(\{0\}) = \{id_{\{0\}}\} \).

We have shown how to get named sets from permutation algebras. Now we have to explain how to get back to \( \text{FSA} \text{l} \text{g}^\pi \). Remember that we have only canonical representatives, and that by identifying all the elements up-to injective relabelling, we have thrown away the global meaning of names. To recover it, the only possibility is to introduce an arbitrary choice sending local names of each element \( q \) to global names, in the form of an injective function \( \rho : \|q\|_N \to \omega \).

**Definition 72** (from \( N \text{Set} \) to \( \text{FSA} \text{l} \text{g}^\pi \)). The functor \( G \) acts on each named set \( N \) as

\[
G(N) = \langle A_N, \{\pi_N\} \rangle
\]

where

\[
A_N = \{\langle q, \rho \circ S_N(q) \rangle \mid q \in Q_N, \rho : \|q\|_N \to \omega \} \\
\pi_N(\langle q, I \rangle) = \langle q, \pi \circ I \rangle
\]

\( G \) acts on each arrow \( F : N \to M \) as

\[
G(F)(\langle q, \rho \circ S_N(q) \rangle) = \langle h_F(q), \rho \circ S_N(q) \circ \Sigma_F(q) \rangle
\]
When mapping back a named set to a permutation algebra, the whole orbit that $q \in Q_N$ represents is reconstructed by the permutation action $\pi_N$. Thus, one has the choice of a permutation $\rho$ which maps the local names $\|q\|_N$ of $q$ to $\omega$, giving a global meaning to them. Composition of the permutation $\rho$ with $S_N(q)$ ensures that the obtained permutation action $\pi_N$ respect the symmetry of $q$ itself. Notice how a morphism follows the mapping $\rho$, by just “carrying it on” using the history of names $\Sigma_F$: the initial and arbitrary choice of $\rho$ uniquely determines the mapping of local names to global ones in elements of the destination of $F$. In particular, the following easy to show lemma shows how to determine the mapping from local to global names in elements of the destination.

**Lemma 73.** It holds that $\rho \circ S_N(q) \circ \Sigma_F(q)$ is equal to $\rho \circ i \circ S_M(h_F(q))$, where $i$ is the injective function (coming from the definition of arrows in Symset) such that $\Sigma_F(q) = i \circ S_M(h_F(q))$.

Also, the following lemma links the support and symmetry of elements of $G(N)$ to the properties of elements of the named set $N$.

**Lemma 74.** The support of an element of $G(N)$ is given by

$$\text{supp}_{G(N)}(\langle q, \rho \circ S_N(q) \rangle) = \rho(\text{dom}(S_N(q)))$$

The symmetry of an element of $G(N)$, on the other hand, is obtained as

$$\mathcal{G}_{G(N)}(\langle q, \rho \circ S_N(q) \rangle)|_{\text{supp}_{G(N)}(\langle q, \rho \circ S_N(q) \rangle)} = \rho \circ S_N(q) \circ \rho^{-1}$$

In words, the symmetry of $\langle q, \rho \circ S_N(q) \rangle$, restricted to its support, is isomorphic to $S_N(q)$. Notice that by theorem 31 the data above is enough to specify the symmetry of $\langle q, \rho \circ S_N(q) \rangle$. 
Chapter 6

Functors for History Dependent Automata

Here we introduce “behavioural” functors for named sets. We start with the most important one: name abstraction. This functor, as we will explain, is responsible for garbage collection of unused names, thus for most of the power of named sets. We will show that the definition is related to the one for permutation algebras by an equivalence of functors. In §6.1, we give the formal definition, and a theorem that establishes a correspondence with the definition in \( \text{FSAlg}_\pi \), exploiting the equivalence result given in [GMM06]. Then, in §6.2, we give an account on how unused names are discarded, by means of some example, to the aim of motivating the main “slogan” we would like to propose in this work: named sets are nominal sets plus garbage collection. Next, we explain how the product is obtained. A very interesting phenomenon can be observed. Intuitively, since names are local, pairing two arbitrary elements of two named sets would make few sense, without explicitly giving a correspondence of names between the two. This was done in all the definitions of history-preserving bisimulation, which is a ternary relation involving two states and a mapping of resources between them. It turns out that the categorical products of two named sets \( N \) and \( M \) contains triples, consisting of an element of \( N \), an element of \( M \) and a name mapping between them, in the form of a multi-coproduct in \( \text{Symset} \) (see section 2.4). In §6.3, we show how the multi-coproduct is defined in \( \text{Symset} \), while in §6.4 we give the actual definition of the product. In contrast, the coproduct has a simple definition, similar to the one in \( \text{Set} \), that we show in §6.6. Finally, we will see how the power set is defined in §6.7. The construction will
enable us in §8.2 to define a modular functor for the semantics of the π-calculus and show that it admits a final coalgebra.

6.1 Name Abstraction

Here we define the endofunctor of abstraction in named sets, which we call \( H \) (standing for “hiding”), with the aid of theorems 34 and 39.

The underlying set \( Q_{H(N)} \) resulting from the action of \( H \) on an object \( N \) is the (disjoint) union of \( Q_N \) itself, representing the orbits of the old elements of definition 36, and a set of pairs \( \langle q \in Q_N, i \in \|q\|_N \rangle \), representing the orbit of the \( i^{th} \) hidden element. Intuitively, \( i \) marks the \( i^{th} \) name of \( q \) as hidden. As we did in theorem 39, the possible values for \( i \) have to be quotiented using the symmetry of \( q \). We set

\[
i \equiv_q j \iff \exists \pi \in S_N(q). \pi(i) = j
\]

and define

\[
Q_{H(N)} = Q_N \cup \{ \langle q, i \rangle \mid q \in Q_N, i \in (\|q\|_N)_{\equiv_q} \}
\]

For readability, in all the following definitions, we implicitly assume the pattern matching on \( q \) and \( \langle q, i \rangle \) to have the additional constraint \( q \in Q_N \), to avoid clashes. The symmetry of elements of the form \( \langle q, i \rangle \) is defined as the subgroup of the symmetry of \( q \) that fixes \( i \) (which we denote with \( gfix(S_N(q), i) \)) according to theorem 34. This symmetry is opportunely restricted in order to exclude \( i \) from the support. We thus define the symmetry as follows:

\[
S_{H(N)}(q) = S_N(q)
\]

\[
S_{H(N)}(\langle q, i \rangle) = gfix(S_N(q), i)_{\|q\|_N \setminus i}
\]

The action of \( H \) on arrows maps \( F : N \to M \) to \( H(F) = \langle h_{H(F)}, \Sigma_{H(F)} \rangle \). A pair \( \langle q, i \rangle \) is mapped by \( h_{H(F)} \) to a pair \( \langle h_F(q), j \rangle \) if and only if \( j \) is mapped by some injection in \( \Sigma_F(q) \) into \( i \), that is, if and only if \( i \) is still present in the destination \( h_F(q) \), according to the history of names \( \Sigma_F(q) \). Formally:

\[
h_{H(F)}(q) = h_F(q)
\]

\[
h_{H(F)}(\langle q, i \rangle) = \begin{cases} 
\langle h_F(q), j \rangle & \text{if } \exists \sigma \in \Sigma_F(q). \sigma(j) = i \\
\{h_F(q)\} & \text{otherwise}
\end{cases}
\]
Notice that in the above definition \( j \) stands for its canonical representative in \( h_{HF}(q) \). For \( h_{HF} \) to be well-defined, we have to show that it respect the equivalence relation on hidden names. This comes from condition (1) in the definition of a named function: for each \( \sigma \in \Sigma_F(q) \) we have \( \Sigma_F(q) = \sigma \circ SM(h_F(q)) \). If there exist \( \sigma' \in \Sigma_F(q) \) and \( j' \neq j \) such that \( \sigma'(j') = i \), then at least a permutation exchanging \( j \) and \( j' \) belongs to \( SM(h_F(q)) \), hence \( j \) and \( j' \) are quotiented by the equivalence relation.

The mapping \( \Sigma_{HF} \) is defined as

\[
\Sigma_{HF}(q) = \Sigma_F(q)
\]

\[
\Sigma_{HF}((q, i)) = \begin{cases} 
\{ \sigma_{|\text{dom}(\sigma)\setminus j} \mid \sigma(j) = i \land \sigma \in \Sigma_F(q) \} & \text{if } h_{HF}((q, i)) = (h_F(q), j) \\
\Sigma_F(q) & \text{otherwise}
\end{cases}
\]

When a hidden name is still present in the destination, the history of names is the subset of \( \Sigma_F(q) \) that sends \( j \) into \( i \), which is then further restricted to \( \text{dom}(\sigma) \setminus j \), so that \( j \) is not mapped at all. This is the same as taking \( \sigma_{|\text{dom}(\sigma)\setminus j} \circ G' \), where \( G' \) is the subgroup of \( SM(h_F(q)) \) fixing \( j \). This follows the correspondence with algebras, and in particular theorem 34. Equivalently, one might augment the codomain of \( \Sigma_{HF} \) to \( \|q, i\|_N + 1 \), thus mapping the fresh name in the destination to a separate name * which is not visible in the source. This is the approach followed in the definitions of history-dependent automata that were given before this thesis work (without noticing that this way to define fresh name generation along morphisms actually comes from the name abstraction functor).

Now we summarise the contents of this section in the following definition.

**Definition 75** (name abstraction). The abstraction functor \( H : \text{NSet} \rightarrow \text{NSet} \) is defined on objects as \( H(N) = (Q_{H(N)}, S_{H(N)}) \), and on arrows as \( H(F) = (h_{HF}, \Sigma_{HF}) \).

**Theorem 76.** \( H \) is a functor.

*Proof.* Let \( F : N \rightarrow M \) and \( G : M \rightarrow O \) be two named functions. The proof that \( H \) preserves identities is trivial, as the proof that \( H(G \circ F) = H(G) \circ H(F) \) in the case of elements of the form \( q \), i.e. without hidden names. We now distinguish two cases. Let \( (q, i) \in H(N) \), \( h_{HF}((q, i)) = (h_F(q), j) \) and \( h_{HF(G\circ F)}((q, i)) = h_G(h_F(q)) \), i.e. the hidden name is discarded by the named function \( G \). Then we have

\[
\exists j \in \|h_F(q)\|_M \exists \sigma \in \Sigma_F(q). \sigma(j) = i \land j \notin \text{Im}(\Sigma_G(h_F(q)))
\]

\[
\Rightarrow h_G(h_F((q, i))) = h_G((h_F(q), j)) = h_G(h_F(q))
\]
This proves that $h_{H(GoF)} = h_{H(G)} \circ h_{H(F)}$. Now we show that the two name mappings are equal. For $I$ set of injective mappings, let $Fix_{j,i}^I$ denote the set
\[ \{ \sigma|_{\text{dom}(\sigma)}: j \to \sigma(j) = i \} \]
From the definition of composition of named functions, and $j \notin \text{Im}(\Sigma_G(h_F(q)))$, we have $Fix_{j,i}^\Sigma F(q) \circ \Sigma_G(h_F(q)) = \Sigma_F(q) \circ \Sigma_G(h_F(q))$.

Now let $\langle q, i \rangle \in H(N)$, $h_{H(F)}(\langle q, i \rangle) = \langle h_F(q), j \rangle$ and $h_{H(GoF)}(\langle q, i \rangle) = \langle h_G(h_F(q)), k \rangle$, thus, the hidden name is preserved in both functions. We have
\[ \exists k \in \| h_G(h_F(q)) \|_O : \exists \rho \in (\Sigma_F(q) \circ \Sigma_G(h_F(q))) \]
\[ \rho(k) = i \land \exists j \in \| h_F(q) \|_M : \exists \sigma \in \Sigma_F(q) . \sigma(j) = i \]
\[ \implies \exists \sigma' \in \Sigma_G(h_F(q)) . \sigma'(k) = j \]
Recall that $i$, $j$ and $k$ represent the equivalence class on hidden names of the definition of $H$. We have that $h_{H(G)}(h_{H(F)}(\langle q, i \rangle)) = \langle h_G(h_F(q)), k \rangle = h_{H(GoF)}(\langle q, i \rangle)$. For the name mappings, due to the definition of composition, and the conditions on named functions, it is easily seen that $Fix_{j,i}^\Sigma F(q) \circ Fix_{k,j}^\Sigma G(h_F(q)) = Fix_{k,i}^{\Sigma F(q) \circ \Sigma G(q)}$.

Finally, we formally show the relationship between $\delta$ and $H$. We regard the following theorem as a correctness result for our definition, with respect to the one for permutation algebras. More details on how we use this result are given in §7.1, after explaining the notion of equivalence of functors. The functor $G$ in the following comes from definition 72.

**Theorem 77.** The two functors $G \circ H$ and $\delta \circ G$ are isomorphic, i.e. there exists a natural transformation $\iota : G \circ H \to \delta \circ G$ such that each component $\iota_N$ is an isomorphism in $\text{FSA}$.

**Proof.** We have to exhibit a natural isomorphism $\iota$ making the following diagram commute for each $K$, $N$ and $M$:

\[
\begin{array}{c c c}
N & G(H(N)) & \overset{\iota_N}{\longrightarrow} \delta(G(N)) \\
\downarrow K & \downarrow G(H(K)) & \downarrow \delta(G(K)) \\
M & G(H(M)) & \overset{\iota_M}{\longrightarrow} \delta(G(M))
\end{array}
\]
The definition of $\iota_N$ is easy:

$$
\iota_N(\langle q, \rho \circ S_{H(N)}(q) \rangle) = \rho_{\delta(G(N))}(\text{old}_{G(N)}(\langle q, S_N(q) \rangle))
$$

$$
\iota_n(\langle \langle q, i \rangle, \rho \circ S_{H(N)}(\langle q, i \rangle) \rangle) = \rho_{\delta(G(N))}(\text{hid}_{G(N)}^i(\langle q, S_N(q) \rangle))
$$

By theorem 30, it suffices to show that $i$ is an isomorphism between canonical representatives of orbits that preserves and reflects the symmetry of elements. The orbits of $\langle q, S_{H(N)}(q) \rangle$ and $\langle \langle q, i \rangle, S_{H(N)}(\langle q, i \rangle) \rangle$ are syntactically distinguished. We take these as canonical representatives.

By theorem 39, the orbits of $\text{old}_{G(N)}(\langle q, S_N(q) \rangle)$ and $\text{hid}_{G(N)}^i(\langle q, S_N(q) \rangle)$ are distinguished in turn. Moreover, all the elements of $G(H(N))$ are covered by the above two cases by definition of $H$, and all elements of $\delta(G(N))$ are in the image of $\iota_n$ because of lemma 38. Thus, $i$ is an isomorphism of (canonical representatives of) orbits. To complete the proof, we have to show that the symmetry is reflected and preserved. By lemma 74, the symmetry of $\langle q, S_{H(N)}(q) \rangle$, restricted to its support, is $S_{H(N)}(q) = S_N(q)$, and the symmetry of $\langle \langle q, i \rangle, S_{H(N)}(\langle q, i \rangle) \rangle$, again restricted to its support, is $gfix(S_N(q, i)_{\|q\|_N})$ by definition 75 and lemma 74 again. By theorem 34 and lemma 74, the symmetry of $\text{old}_{G(N)}(\langle q, S_N(q) \rangle)$ restricted to its support is $S_N(q)$ and the symmetry of $\text{hid}_{G(N)}^i(\langle q, S_N(q) \rangle)$ is $gfix(S_N(q, i)_{\|q\|_N})$.

We now have to show that the isomorphism is natural, i.e. it commutes with arrows (it suffices to show this for canonical representatives). For elements without hidden names, the proof is trivial. For an element $\langle \langle q, i \rangle, S_{H(N)}(\langle q, i \rangle) \rangle$, we have two cases: either $\exists \sigma \in \Sigma_K(q). \sigma(j) = i$ or not. In the first case, let $\Sigma_K(q) = \rho \circ S_M(h_K(q))$. We have

$$
\iota_M(G(H(K))(\langle \langle q, i \rangle, S_{H(N)}(\langle q, i \rangle) \rangle))
$$

$$
=\iota_M(\langle \langle h_K(q), j \rangle, \rho \circ gfix(S_M(q, j)_{\|q\|_M}) \rangle)
$$

$$
=\rho_{\delta(G(M))}(\text{hid}_{G(M)}^i(\langle q, S_M(q) \rangle))
$$

For the way $\rho$ is introduced, see lemma 73. We also have

$$
\delta(G(K))(\iota_N(\langle \langle q, i \rangle, S_{H(N)}(\langle q, i \rangle) \rangle))
$$

$$
=\delta(G(K))(\text{hid}_{G(N)}^i(\langle q, S_N(q) \rangle))
$$

$$
=\rho_{\delta(G(M))}(\text{hid}_{G(M)}^i(\langle q, S_M(q) \rangle))
$$

The other case is similar. □
6.2 Garbage Collection

In this section we give a semi-formal account of the garbage collection property of named sets and their coalgebras, history-dependent automata. When we talk about the semantics of the $\pi$-calculus in permutation algebras and presheaves, we refer, respectively, to the one that we give in §4.2, and to its translation via the equivalence between the two categories. It is important to clarify that the examples that we depict in this section are not subobjects of the final coalgebra, but rather the operational semantics of agents that is directly obtained by rules. We present some very simple cases where this semantics is infinite, while it is finite using HD-automata. It should be clear that this difference does not hold (due to full abstractness of the semantics) in the final coalgebra, where all the infinite different states are identified. However, there is no standard way to compute the final coalgebra starting from an infinite set of states. Using HD-automata, thus starting from a finite set of states in various cases, computing the final coalgebra is done by iteration along the terminal sequence.

Consider the definition of $h_{H\mathcal{F}}((q, i))$ in the abstraction functor for named sets and observe that, when the hidden name $i$ is discarded along a morphism, the resulting element is just $q$. This introduces garbage collection, allowing the semantics to reuse old states whenever a fresh name is discarded. We now attempt to give an intuition of this fact by the means of two examples in $\pi$-calculus. We ignore labels of transitions, since these do not contribute to the intuition, and we omit to denote the power set, because both systems are deterministic. When representing HD-automata, we draw the backward mappings of names, together with supports of states, side by side with the transition function. Since the symmetry of both the agents we present is just $\{id\}$, a backward mapping $\Sigma$ is represented by a single function.

Consider the agent $P(1) = (\nu x)\text{I}x.P(1)$. Even though it has no memory of the past, thus after just one step there are no more free names to be discarded, the operational semantics of the system in $\text{FSAAlg}^{\pi}$ reaches all the (infinite, countable) elements in the orbit of $P(1)$. As we said above, this is not the case in the final semantics where $P(1)$ does just a simple cycle. However, the semantics that is derived by the rules is infinite, and as a result it becomes impossible, without recurring to a finite representation, to do model-checking or minimisation of the obtained model. Figure 6.1 depicts a sketch of the permutation algebra semantics of $P(1)$, compared to its finite representation as an HD-automaton in figure 6.2, which is a simple loop. The transition (named) function
6.2. GARBAGE COLLECTION

State Support

$$(\nu x)\overline{1}x.P(1) \quad \text{supp} = \{1\}$$

$$(\nu x)\overline{2}x.P(2) \quad \text{supp} = \{1\}$$

$$(\nu x)\overline{3}x.P(3) \quad \text{supp} = \{1\}$$

$$\cdots \quad \text{supp} = \{1\}$$

Figure 6.1: Infinite states in $\text{FSAlg}^{\pi}$

State Support

$$(\nu x)\overline{1}x.P(1) \bigcup_{h_{tr}} \bigcup_{\Sigma_{tr}} \{1\}$$

Figure 6.2: HD-automaton having finite states

$tr = \langle h_{tr}, \Sigma_{tr} \rangle$ acts on $P(1)$ (intended as the canonical representative of its whole orbit) as $h_{tr}(P(1)) = \{P(1)\}$, $\Sigma_{tr}(P(1)) = \{id\{1\}\}$.

Now let $R(1) = (\nu x)\overline{1}x.R(x)$. Consider a presheaf semantics for the $\pi$-calculus. On the left of each state, we draw in Fig. 6.3 the least stage (object of the base category) in which the state is found at all (the categorical support of the element, as it is called in [GMM06]), and the stage in which the coalgebra is applied to it. Recall that, given a presheaf $T$, the functor $\delta : \text{Set}^{I} \rightarrow \text{Set}^{I}$ is defined on objects as $\delta(T)(X) = T(X \oplus 1)$.

To distinguish the different instances, in the successive applications of the coproduct, of the only element $* \in 1$ (the final object of $I$), we denote it with $*, '*, ''* \text{ and so on.}$

In Fig. 6.4, we can find the HD-automata semantics of $R(1)$. The HD-automata semantics is now made up of two states, since in the first step the free name $a$ has to disappear. The transition function is $h_{tr}(R(1)) = \{(R(1), 1)\}$, thus hiding name 1, and $h_{tr}((R(1), 1)) = \{(R(1), 1)\}$. We have the name mapping $\Sigma_{tr}(R(1)) = \Sigma_{tr}((R(1), 1)) = \{\emptyset\}$, the empty name mapping. This is required since the support of the destination is empty.
### Support Stage in the coalgebra State

<table>
<thead>
<tr>
<th>Support</th>
<th>Stage in the coalgebra</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>{1}</td>
<td>((\nu x)I.x.R(x))</td>
</tr>
<tr>
<td>{\ast}</td>
<td>{1} + 1 = {1, \ast}</td>
<td>((\nu x)\overline{I}.x.R(x))</td>
</tr>
<tr>
<td>{\ast'}</td>
<td>{1, \ast} + 1 = {1, \ast, \ast'}</td>
<td>((\nu x)\overline{I}'.x.R(x))</td>
</tr>
<tr>
<td>{\ast''}</td>
<td>{1, \ast, \ast'} + 1 = {1, \ast, \ast', \ast''}</td>
<td>((\nu x)\overline{I}''.x.R(x))</td>
</tr>
</tbody>
</table>

... 

Figure 6.3: Infinite states in \(\text{Set}^I\)

### State Support

\[
\begin{align*}
(\nu x)I.x.R(x) & \quad \{1\} \\
\uparrow_{htr} & \\
\langle (\nu x)I.x.R(x), [1] \rangle & \quad \emptyset
\end{align*}
\]

Figure 6.4: HD-automaton having finite states
What both permutation algebras and functors in \( \text{Set} \) lack, is a mechanism to discard unused names using a quotient operation: on orbits, in the case of permutation algebras, on isomorphic categorical supports, in the case of presheaves.

These examples should explain what we mean with \textit{locality} of names. In particular, notice how the backwards name mappings of named functions trace the history of names along morphisms, allowing the semantics to reuse the same state with different names. One might wonder if this is just a trick and, in the end, one has to perform an “unfolding” of the HD-automata semantics into the ordinary LTS semantics to implement algorithms such as bisimulation checking or model checking. Results in [FMT05a] for minimisation and bisimulation checking, and work in progress on the model checking side, show that this is not necessary, and the model can be used “as is” to verify systems up-to garbage collection.

### 6.3 Multi-Coproducts in Symset

Consider two objects \( \Phi_1 \) and \( \Phi_2 \) of \textit{Symset}, and a cospan

\[
\begin{array}{c}
\Phi_1 \\
\downarrow^f \\
\Phi \\
\downarrow^g \\
\Phi_2
\end{array}
\]

between them. We are interested in using such a pair of functions as a symmetric notion of mapping of names between \( \Phi_1 \) and \( \Phi_2 \), allowing some names to be identified, and others not to be mapped at all. It should be clear that the middle object is of no interest to us by itself, but only as a representation of the common and distinguished names of the two objects. This has two consequences: first, we are not interested in names of \( \Phi \) that are outside the image of \( f \) and \( g \); second, we are not interested in distinguishing two isomorphic cospans, because the obtained name mapping is the same. The multi-coproduct construction contains minimal cospans quotiented by isomorphism, thus we will employ it to represent our name mappings.

First, we observe that \textit{Symset} does not have coproducts since its full subcategory \( \text{I} \) does not have them in turn. In the following we denote with 1 and 2 (without bold face) a one element and a two elements subsets of \( \omega \), respectively. Consider the objects \( 1 = \{id_1\} \) and \( 2 = \{id_2\} \), and the cocones

\[
L_1 = \begin{array}{c}
1 \\
\downarrow^{id_1} \\
1
\end{array}
\]

\[
L_2 = \begin{array}{c}
1 \\
\downarrow^{true} \\
2 \\
\downarrow^{false} \\
1
\end{array}
\]
over the same diagram $D = (1, 1)$. The arrows \textit{true} and \textit{false} select different elements of the set 2. It is easily seen that due to injectivity, there is no single limiting cocone that fits both diagrams. However, \textit{Symset} has multi-coproducts of finite diagrams (here we only state the case of two objects for simplicity).

**Theorem 78.** The multi-coproduct of $(\Phi_1, \Phi_2)$ in \textit{Symset} is given by the set of cospans of the form $(\text{id}_{\Phi_1} : \Phi_1 \rightarrow \Phi, \text{id}_{\Phi_2} : \Phi_2 \rightarrow \Phi)$ such that

1. $\text{dom}(\Phi) = \text{Im}(\text{id}_{\Phi_1}) \cup \text{Im}(\text{id}_{\Phi_2})$

2. $\Phi$ is the greatest permutation group over $\text{dom}(\Phi)$ such that $\text{id}_{\Phi_1}$ and $\text{id}_{\Phi_2}$ respect the conditions of definition 61 to make them arrows of \textit{Symset} quotented by cospans isomorphism.

Informally, names in $\Phi$ are meaningful since they are in the image of at least one $\text{id}_{\Phi_i}$, while the target of an arbitrary cocone may contain extraneous names that do not have a history according to any $f_i$. The condition on maximality of the symmetry ensures existence of the unique mediating morphism.

The proof requires the following lemma, that is an immediate consequence of the second condition in the definition of arrows of \textit{Symset} (see definition 61).

**Lemma 79.** Let $F : \Phi_1 \rightarrow \Phi_2$ be an arrow of \textit{Symset}, and let $i \in \text{dom}(\Phi_2) \setminus \text{Im}(F)$. Then, there is no $\rho \in \Phi_2$ such that $\rho(i) \in \text{Im}(F)$.

Now we can prove theorem 78.

\textit{Proof.} Let $\text{id}_{\Phi_1} = i_1 \circ \Phi_1, \text{id}_{\Phi_2} = i_2 \circ \Phi_2$. For the first condition, suppose there exists a name $n \in \text{dom}(\Phi)$ such that $i \notin \text{Im}(\text{id}_{\Phi_1}) \cup \text{Im}(\text{id}_{\Phi_2})$. Let $\Phi'$ denote the subgroup obtained by restricting all the permutations in $\Phi$ to $\text{Im}(\text{id}_{\Phi_1}) \cup \text{Im}(\text{id}_{\Phi_2})$, which is indeed a subgroup by lemma 79 above. Let $\iota : \text{dom}(\Phi') \rightarrow \text{dom}(\Phi)$ be the inclusion in $I$. Then $\iota \circ \Phi'$ is an arrow of \textit{Symset}, and both $\text{id}_{\Phi_1}$ and $\text{id}_{\Phi_2}$ factor through it. $\iota$ is not an isomorphism, hence the cospan $\langle \text{id}_{\Phi_1}, \text{id}_{\Phi_2} \rangle$ is not an element of the multi-coproduct: there is another non-isomorphic cospan that commutes with it via the arrow $\iota \circ \Phi'$.

For the second condition, if $\Phi$ is not the greatest possible subgroup, then there is $\Phi'$ such that $i_1 \circ \Phi_1$ can be viewed as an arrow of type $\Phi_1 \rightarrow \Phi'$, and similarly $i_2 \circ \Phi_2$ can be viewed as an arrow of type $\Phi_2 \rightarrow \Phi'$. Then, we have a cospan with $\Phi'$ as a target, and a commuting arrow from $\Phi'$ to $\Phi$, namely $\text{id}_{\text{dom}(\Phi)} \circ \Phi'$. This arrow is not an isomorphism, hence the cospan $\langle \text{id}_{\Phi_1}, \text{id}_{\Phi_2} \rangle$ is not an element of the multi-coproduct. \hfill $\square$
6.4 The Categorical Product of Named Sets

The categorical product is often exploited to generalise the set-theoretical notion of relation, and define it in categories different than Set. In named sets the product exemplifies locality of names, i.e. that identity of names has to be established when two elements are put in a relation. As a fundamental consequence, bisimulation of HD-automata is a ternary relation, comprising two states and a name mapping between them.

The “non-standard” notion of bisimulation dating back to causal automata [MP97] is thus recovered in a purely coalgebraic framework.

Getting into detail, a relationship between two objects \( \phi_1 \) and \( \phi_2 \) of \( \text{Symset} \) is represented here by a cospan, i.e. a diagram of the form \( \phi_1 \overset{\text{in}_1}{\longrightarrow} \phi \overset{\text{in}_2}{\longrightarrow} \phi_2 \). Names in \( \text{Im} (\text{in}_1) \cap \text{Im} (\text{in}_2) \) are those that are related, while the others are kept distinct. Many cospan is equivalent from our point of view, because they are isomorphic cocones, and many others contain redundant names in the middle object \( \phi \). Employing elements of a multi-coproduct in \( \text{Symset} \), one gets rid of these unnecessary cospan. The categorical product of named sets \( N \) and \( M \) is made up of triples containing an element of \( Q_N \), an element of \( Q_M \), and an element of a multi-coproduct in \( \text{Symset} \).

**Theorem 80.** The categorical product of \( N \) and \( M \) is given by the named set \( \langle Q,S \rangle \) where

\[
Q = \{ \langle n,m,\langle \text{in}_1,\text{in}_2 \rangle \rangle \mid n \in Q_N \land m \in Q_M \land \langle \text{in}_1,\text{in}_2 \rangle \in \text{MCL}(\langle S(n),S(m) \rangle) \}
\]

\[
S(\langle n,m,\langle \text{in}_1,\text{in}_2 \rangle \rangle) = \text{cod} (\text{in}_1) = \text{cod} (\text{in}_2)
\]

The projections \( \pi_1 = \langle h_1,\Sigma_1 \rangle \) and \( \pi_2 = \langle h_2,\Sigma_2 \rangle \) are defined as

\[
h_1(t) = n \]
\[
h_2(t) = m \]
\[
\Sigma_1(t) = \text{in}_1 \]
\[
\Sigma_2(t) = \text{in}_2 \]

Proof. For each named set \( C \), and arrows \( F:C \rightarrow N, G:C \rightarrow M \), we define the unique morphism from \( C \) to \( N \times M \) as \( \langle h,\Sigma \rangle \), where \( h(q \in Q_C) = \langle h_F(q), h_G(q), \langle \text{in}_1,\text{in}_2 \rangle \rangle \), and \( \Sigma(q) = u \). Here \( \langle \text{in}_1,\text{in}_2 \rangle \in \text{MCL}(\langle S_N(h_F(q)),S_M(h_G(q)) \rangle) \), with \( u \) unique morphism commuting with the cocone \( \langle \Sigma_F(q),\Sigma_G(q) \rangle \) in \( \text{Symset} \). The set-theoretical part of the construction is unique since it corresponds to the set theoretical product. The name mappings are also unique, from uniqueness of the multi-coproduct. \( \square \)
The middle object \( \phi \) represents the symmetry (and the names) of an element of the categorical product. Co-universality of the limiting cocone ensures that \( \phi \) acts on \( \text{Im}(in_1) \cap \text{Im}(in_2) \) as the intersection of the symmetries of \( n \) and \( m \), and on names in \( \text{Im}(in_1) \setminus \text{Im}(in_2) \) and \( \text{Im}(in_2) \setminus \text{Im}(in_1) \), respectively, as the symmetry of \( n \) or of \( m \). This corresponds to the properties of the product in \( \text{FSA}lg^{\pi} \) that we have given in theorem 52 and corollary 53.

An intuition can also be given starting from permutation algebras, as follows: for \( \langle A, \{ \pi_A \} \rangle \) and \( \langle B, \{ \pi_B \} \rangle \) permutation algebras, \( a \in A \) and \( b \in B \), having orbits \( \{ \pi_A(a^o) \mid \pi \in \text{Autf} \} \) and \( \{ \pi_B(b^o) \mid \pi \in \text{Autf} \} \), an element of the product can be any pair \( \langle \pi'_A(a^o), \pi''_B(b^o) \rangle \).

The maps \( \pi' \) and \( \pi'' \) give rise to \( in_1 \) and \( in_2 \) through the equivalence between \( \text{FSA}lg^{\pi} \) and \( \text{NSet} \).

The action of the (bi)functor on arrows is induced by the unique morphism of the limit construction (see proof of theorem 80), however we can define it explicitly as follows.

**Definition 81 (product).** The categorical product sends each pair of arrows \( F : N \to N' \) and \( G : M \to M' \) into

\[
F \times G = \langle h, \Sigma \rangle
\]

For \( t = \langle n, m, (in_1, in_2) \rangle \in Q_N \), let

\[
\langle in'_1, in'_2 \rangle \in \text{MCL}(\langle S_N(n), S_M(m) \rangle)
\]

be the unique element of the multi-coproduct commuting with the cocone

\[
\langle in_1 \circ \Sigma_F(n), in_2 \circ \Sigma_G(m) \rangle
\]

and \( u \) the associated unique arrow. Then we define

\[
h(t) = \langle h_F(n), h_G(m), \langle in'_1, in'_2 \rangle \rangle
\]

\[
\Sigma(t) = u
\]

The underlying functions \( h_F \) and \( h_G \) are paired as in \( \text{Set} \). The injections are also paired, and composed with the name mappings \( \Sigma_F(n) \) and \( \Sigma_F(m) \). This composition results in a cocone that is not guaranteed to belong to a multi-coproduct. Thus, the multi-coproduct is used to find a canonical form for this cocone. The name mapping associated to the pairing is then the unique mapping obtained from the multi-coproduct, as explained in Fig. 6.5.
6.4. THE CATEGORICAL PRODUCT OF NAMED SETS

The following example illustrates how the history of names along morphisms keeps morphisms distinct (in this case, the two projections of the categorical product), even though orbits are quotiented.

**Example 82.** Continuing from examples 68 and 71, we now define the categorical product \( \langle Q, S \rangle \) of the named set of natural numbers with itself. The multi-coproduct in \( \text{Symset} \) of the object \( \{ \text{id}_{\{0\}} \} \) with itself is represented by the two cospans \( \langle \{f\}, \{f\} \rangle \) and \( \langle \{g\}, \{g'\} \rangle \) where \( \{f\} : \text{id}_{\{0\}} \to \text{id}_{\{0\}} \), with \( f(0) = 0 \), \( \{g\} : \text{id}_{\{0\}} \to \text{id}_{\{0,1\}} \) with \( g(0) = 0 \), and \( \{g'\} : \text{id}_{\{0\}} \to \text{id}_{\{1\}} \), with \( g(0) = 1 \). This corresponds to say that two names in the product \( \omega \times \omega \) are either distinct or equal.

Thus, the underlying set \( Q \) consists of the two tuples

\[
q_1 = (0,0, \{f\}, \{f\}) \\
q_2 = (0,0, \{g\}, \{g'\})
\]

The symmetry is defined as \( S(q_1) = \{ \text{id}_{\{0\}} \} \) and \( S(q_2) = \{ \text{id}_{\{0,1\}} \} \). The two projections \( \pi_1 = \langle h_1, \Sigma_1 \rangle \) and \( \pi_2 = \langle h_2, \Sigma_2 \rangle \) are defined by \( h_1(q_1) = h_2(q_1) = h_1(q_2) = h_2(q_2) = 0 \), and

\[
\Sigma_1(q_1) = \Sigma_2(q_1) = \{f\} \\
\Sigma_1(q_2)(0) = \{g\} \\
\Sigma_2(q_2)(0) = \{g'\}
\]

To conclude this section, we introduce a very interesting point: to characterise the product, we employ the multi-coproduct in \( \text{Symset} \). Actually, Diers proved in [Die76] that existence of multi-limits in a category is equivalent to existence of limits in its free coproduct completion. As it is shown in [Sta07], the free coproduct completion of the category \( \text{Coset} - \text{Act} \) defined therein corresponds to the category of named sets and named functions. Indeed, our definition of named sets is very similar to those in [Sta07],
so existence of the product is directly related to the result of Diers. This is promising in view of generalising the construction of named sets from permutation algebras to categories of algebras with more structured signatures (equivalently, to categories of presheaves with richer index categories than $I$). Further study is required on such “categories with local interfaces” (we mention this topic in the conclusions of the thesis), however the idea that, if two systems have local interfaces, then to compose them we have to establish a map that binds their interfaces, seems to be preserved by the generalisation.

## 6.5 Locality of Names, Revisited

A digression is now due on the matter of locality of names. Recall from §5.3 that backwards name mappings come out after quotienting orbits of permutation algebras by injective relabelings. Thus, most of the power of named sets is just a consequence of a design choice that discards the global meaning of names. Notice that the idea of keeping track of the history of names using a backwards injective relabelling is common in computer science. As an example, consider memory allocation in programming languages with dynamic memory: the process memory is handled as a data structure indexed by memory locations (the heap). Whenever a fresh memory block allocation is requested, a free memory area which is large enough has to be found. This process may involve a compaction of the heap, typically to gain more continuous space. This is done by a relocation function. If one thinks of addresses of all memory blocks as names that are attached to each block, the relocation function becomes nothing more than an injective renaming, and the explanation in named sets terminology of the relocation machinery is that names attached to memory blocks do not have a global and immutable meaning, but rather their meaning is established in each step of the transition function of the memory allocation system. Notice that also in this case there exists, in principle, another semantics that handles names as global: it is the abstract semantics of the programming language itself, or, one might say, the corresponding nominal sets semantics.

Thus, a well-understood common practice in computing can be explained in categorical terms by a quotient operation and an equivalence result. However, memory allocation is neither the only common manifestation of locality of names in everyday computing, nor the only one that is explained by a construction in named sets. Consider the following example that gives an intuition on the mapping of names of the categorical
product: variables or function symbols in programming languages. These names usually
have only a local meaning, since they have been defined in the scope of the program itself,
and are not known outside of it. Think now of the operation of composing two program
fragments, by syntactically pasting a fragment into the other one. All the variables of
the involved parts should in principle be renamed in order to avoid a clash. Thus, one
might be tempted to say that variables are bound, $\alpha$-convertible names. However, the
rules of $\alpha$-conversion alone would be too restrictive: it may happen that some variables
(for example, library functions, or variables that have the same meaning in both frag-
ments) actually have to clash, and the laws of $\alpha$-conversion would prohibit this. Instead,
a correspondence of names has to be established by appropriately renaming local sym-
bols before pasting, in order to avoid a clash, and letting the names of common symbols
overlap.

Notice that even the library symbols do not have a well defined, completely global
meaning: library functions are just “more global” in the sense that they belong to a more
general context. Different systems and libraries may have the same names for different
operations, requiring renaming of symbols that had to be left untouched in previous
steps, as the context enlarges. For similar reasons, most programming languages pair
the possibility to import library declarations with a syntactic construct of renaming of
library symbols, to allow importing different libraries whose names might overlap. The
categorical product of named sets that we introduced provides a formal approach to this
kind of composition operations with local names.

A similar global/local names issue is often encountered in distributed systems: con-
sider two isolated networks, with each node having a name, locally known to the other
machines of its neighbourhood. When one wants to link the two networks to each other,
a binding machinery is needed to establish which names of one network are not shown in
the other one, and which are bound to a name in the common domain. Similar problem-
atics where encountered in CHARM [CMR94], an abstract machine based on rewriting,
that models distributed systems consisting of a collection of processes sharing a subset
of their variables, and keeping distinct their private ones.

### 6.6 The Coproduct

Compared to the other functors that we have defined, the coproduct has a fairly simple
structure, just inherited from the set-theoretical definition. In the following we denote
with \([f; g]\) the copairing of \(f\) and \(g\), and with \textit{join} an operation sending elements of \(a + b\) into \(a \cup b\).

**Theorem 83.** Given two named sets \(N = \langle Q_N, S_N \rangle\) and \(M = \langle Q_M, S_M \rangle\), their coproduct is given by \(\langle Q_N + Q_M, \text{join} \circ [S_N; S_M] \rangle\), and the injections are the set-theoretical ones, with identity name mappings.

**Definition 84** (coproduct). The coproduct is a bifunctor acting on each pair of arrows \(F = \langle h_F, \Sigma_F \rangle\), \(G = \langle h_G, \Sigma_G \rangle\) as \(\langle h, \Sigma \rangle\) where \(h = [h_F; h_G]\) and \(\Sigma = \text{join} \circ [\Sigma_F; \Sigma_G]\).

### 6.7 The Countable Power Set

In this section, we present the countable power set functor \(P_\omega(-)\). As in the case of the product, a multi-coproduct is used to establish a relationship between names of the elements of each subset. One might expect an element of \(P_\omega(N)\) to be a set \(p\) of pairs \(\langle q_i, \text{in}_i : S_N(q) \rightarrow \phi \rangle_{i \in I}\), with \(I\) finite or countable. This approach presents two difficulties. First, one has to consider the implicit symmetry due to \(p\) being a set, not a tuple. Symmetries cannot grow along the \(\text{in}_i\) as they are morphisms of \(\text{Symset}\), thus \(\phi\) cannot contain this symmetry, as it is the target of all the \(\text{in}_i\): locally, each \(q_i\) might even not have symmetries at all. Therefore, differently from the categorical product, \(\phi\) is not the symmetry of \(p\). Instead, it should be \textit{completed} with additional permutations in order to play this role. Second, to obtain equivalence with respect to the functor in \(\text{FSAAlg}^\pi\), we need to describe canonical representatives of orbits of \(P_\omega(A)\) for each permutation algebra \(A\). By looking at example 56, it should be clear that a \textit{finite} object \(\phi\) may not be sufficient.

Our definition is based on the following idea: we first give up on the finite support requirement on \(\phi\), and then “recover” it by selecting only those subsets that actually are finitely supported. Consider an extension of \(\text{Symset}\): the category \(\text{Symset}^\omega\) whose objects are groups of permutations \(\{\phi \in \text{Grp}(S) \mid S \in P_\omega(\omega)\}\), having a \textit{countable} domain. It is easily shown that it is has multi-coproducts of countable diagrams, and that \(\text{Symset}\) is a full subcategory of it.

**Definition 85** (power set). We say that \(\phi \in |\text{Symset}^\omega|\) is finitely supported if there exists a finite supporting set \(S \in P_{\text{fin}}(\omega)\) such that \(\phi_{|\text{dom}(\phi)\setminus S}\) contains all the possible permutations over \(\text{dom}(\phi)\setminus S\). In this case, we define \(\text{fin}(\phi) \in |\text{Symset}|\) as \(\phi_{|T}\), where \(T\) is the smallest finite supporting set.
For \( N \) named set, let \( P(N) \) denote all the sets of pairs \( \{ \langle q_i, i \in I \rangle \mid q_i \in Q_N \land i \in I \} \) with \( \phi \in |\text{Symset}^\omega| \) and \( I \subseteq \omega \). We now define the completion we mentioned above, to represent the unordered nature of the power set.

**Definition 86** (completion). Let \( N \) be a named set, \( p \in P(N) \). The completion of \( p \) is an object of \( \text{Symset}^\omega \), defined as the greatest permutation group \( p^c \) over \( \text{dom}(\phi) \) such that \( p = \{ \langle q_i, \rho \circ i \rangle \mid \rho \in p^c \land \langle q_i, i \rangle \in p \} \).

The definition of the countable power set follows. The action of the functor on arrows employs the multi-coproduct in a similar way to definition 81.

**Definition 87** (countable power set). The countable power set is defined as

\[
P_\omega(N) = \langle Q, S \rangle
\]

where \( Q \) is the greatest subset of \( P(N) \) such that for each \( p = \{ \langle q_i, i \rangle \mid i \in I \} \in Q \), the cocone \( \langle i \rangle_{i \in I} \) is in \( \text{MCL}(\langle S_N(q_i) \rangle_{i \in I}) \) and \( p^c \) is finitely supported. The symmetry is defined as

\[
S(p) = \text{fin}(p^c)
\]

Consider an arrow \( F = \langle h_F, \Sigma_F \rangle \) from \( N \) to \( M \). We define the action of the power set on arrows \( P_\omega(F) = \langle h, \Sigma \rangle \) as follows. Let

\[
p = \{ \langle q_i, i \rangle \mid i \in I \} \in Q_{P_\omega(N)}
\]

Let \( \langle i \rangle_{i \in I} \) and \( u \) denote the unique element of \( \text{MCL}(\langle S_M(h_F(q_i)) \rangle_{i \in I}) \) commuting with

\[
\langle i \circ \Sigma_F(q_i) \rangle_{i \in I}
\]

and the associated unique morphism, respectively. Then we define

\[
h(p) = \{ \langle h_F(q_i), i \rangle \mid i \in I \}
\]

\[
\Sigma(p) = u \circ S_{P_\omega(M)}(h(p)) \quad \text{(composed as sets of functions)}
\]

To conclude this section, we state an isomorphism result analogous to that of theorem 77. We recall that the functor \( G \) in the following comes from definition 72.

**Theorem 88.** Let \( P_F^* \) denote \( P_\omega \) in \( \text{FSAlg}^* \) and \( P_N^* \) denote the corresponding functor on named sets. The two functors \( G \circ P_N^* \) and \( P_F^* \circ G \) are isomorphic as functors.
**Proof.** Starting from a permutation algebra $A = \langle A, \{\pi_A\} \rangle$, we have to find canonical representatives, and their symmetry, for $P_\omega(A)$ in terms of those of $A$. Let $p \in P_\omega(A) = \{a_i \mid i \subseteq \omega\} = \{\pi^i_A(a^i_o) \mid i \subseteq \omega\}$. First, the choice of the various $\pi^i$ is implicitly quotiented with the symmetry of $a^i_o$ in a permutation algebra, since $\sigma \in G_A(a^i_o) \Rightarrow (\pi^i \circ \sigma)_A(a^i_o) = \pi^i_A(a^i_o)$. Hence, all the $\pi^i_A$ may be regarded as arrows of $\text{Symset}^\omega$ as $\pi^i_A \circ G_A(a^i_o)_{|\text{supp}_A(a^i_o)}$. Next, we look at how permutations act on the whole $p$. We have $\tilde{\pi}_{P_\omega(A)}(p) = \{\tilde{\pi}(\pi^i_A(a^i_o)) \mid i \subseteq \omega\} = \{(\tilde{\pi} \circ \pi^i)_A(a^i_o) \mid i \subseteq \omega\}$. The orbit of $p$ is thus obtained using all the possible cocones over $G_A(a^i_o)_{|\text{supp}_A(a^i_o)}$ in $\text{Symset}^\omega$. The support of $p$ cannot contain names outside the image of at least one $\pi^i$, since no $\tilde{\pi}$ can change $p$ acting only on such names. Therefore, a canonical representative of the orbit of $p$ contains all the $a^i_o$ paired with an arrow of $\text{Symset}^\omega$, so that the resulting cocone is a canonical representative of all the equivalent ones and contains no extraneous names. This is the definition of multi-coproduct in $\text{Symset}^\omega$. The symmetry of $p$ is given by $\{\tilde{\pi} \in \text{Autf} \mid \{(\tilde{\pi} \circ \pi^i)_A(a^i_o) \mid i \subseteq \omega\} = p\}$, that corresponds to the definition we gave in $\text{NSet}$. Also, the condition of being finitely supported correspond precisely to definition 85.

For the action of the functor on arrows, observe that for $f : A \to B$ in $\text{FSAlg}^\ast$, the action of $P_\omega(f)(p)$ may return an element which is not a canonical representative of its orbit, however it always has the form $p' = \{\pi^i_B(f(a^i_o)) \mid a_i \in p\}$ because of properties of morphisms. The unique morphism obtained from the multi-coprodct is then the backward name mapping of the action of the functor on $f$ in $\text{NSet}$ since it makes name mappings commute (it is the identity in the case when $p'$ is canonical). \qed
Chapter 7

The Final Coalgebra Theorem

The idea that we present in this chapter is to lift a category of coalgebras from a domain to another exploiting a pair of equivalent functors. There is a close link between a duality and an equivalence, that is, an equivalence between $A$ and $B$ is a duality between $A$ and $B^{op}$. This methodology is thus intrinsically similar to the grounds of the Stone-type duality for coalgebras and modal logics that was developed in [BK05] and related bibliography. There, one is interested in finding a suitable category of algebras that provide a logical representation of a class of systems, using a pair of dual functors. In our case, the focus is on relating a category with pleasant algorithmic properties (coalgebras over $\text{NSet}$) to categories having well-known mathematical properties (such as coalgebras over $\text{FSAlg}^\pi$, or equivalently on “well behaved” presheaf categories like the Schanuel topos).

On one hand, this gives a clean and understandable way to reason about, and prove properties of, HD-automata by looking at them at a more abstract level. On the other hand, the equivalence guarantees that semantics expressed using more abstract formalisms has a counterpart that can be implemented in a wide range of cases. For example, the ad-hoc approach followed for the $\pi$-calculus in [FMT05a] could as well be generalised to other name passing calculi whose semantics is expressible as a coalgebra in $\text{FSAlg}^\pi$, or equivalently $\text{Set}^1$.

7.1 Lifting Equivalences to Categories of Coalgebras

We introduce the notion of pair of equivalent endofunctors.
Definition 89 (equivalent functors). Given an equivalence \((F, G, \eta, \epsilon)\) between the categories \(A\) and \(B\), two endofunctors \(T : A \to A\) and \(S : B \to B\) are equivalent if there exists a natural isomorphism \(k : F \circ T \to S \circ F\), or (equivalently) \(l : G \circ S \to T \circ G\).

\[ T \quad F \quad A \quad G \quad B \quad S \]

Figure 7.1: Equivalent Functors

The diagram in Fig. 7.1 depicts the situation we are interested in. It is easy to show that composition of functors and subfunctors preserves equivalence, which we state in the following lemma.

Lemma 90. If \(F\) and \(G\) are equivalent respectively to \(F'\) and \(G'\), respectively, then \(F \circ G\) is equivalent to \(F' \circ G'\). Moreover, for each subfunctor \(S\) of \(F\), there is a corresponding equivalent functor \(S'\) which is a subfunctor of \(F'\).

The notion of equivalence of functors allows us to derive our final coalgebra theorem as follows.

Theorem 91. If \(T\) and \(S\) are equivalent, \(T\) is accessible if and only if \(S\) is.

Proof. Consider a \(\lambda\)-filtered colimit diagram \(D\) in \(A\) and assume \(T\) accessible. The diagram \(F(D)\) is a \(\lambda\)-filtered colimit since equivalences of categories preserve limits and colimits. By hypothesis \(T(D)\) is a \(\lambda\)-filtered colimit, and so is \(F(T(D))\). The diagram \(S(F(D))\) is isomorphic to \(F(T(D))\), thus it is a \(\lambda\)-filtered colimit in turn. \(\Box\)

Definition 89 can be made slightly more precise by the means of the following lemma.

Lemma 92. In the case of definition 89, the definitions of \(l\) and \(k\) are tied to each other: for each \(a\) it holds

\[ l_a = \eta_{T(G(a))}^{-1} \circ G(k_{G(a)}^{-1}) \circ G(S(\epsilon_a^{-1})) \]

Proof. Just observe that the composition of natural isomorphisms is a natural isomorphism, and that the types are correct, hence the definition of \(l\) is a natural isomorphism of the correct type in turn. \(\Box\)
For what it concerns to us, theorem 91 ensures that both $T$ and $S$ admit a final coalgebra. However, we can be more precise, and exhibit an equivalence result between the associated categories of coalgebras. This ensures that whenever there exists a fully abstract coalgebraic semantics for the functor $T$ in the category $A$, the same happens for the functor $S$ in the category $B$. Thus, we can specify our coalgebras e.g. in nominal sets, where it is easy (see §4.2 as an example), and then be confident that a corresponding semantics can be obtained in named sets (as we do in §8.2).

**Theorem 93.** Let $T$ and $S$ be equivalent functors via $(F, G, \eta, \epsilon)$, and let $k : F \circ T \to S \circ F$, $l : G \circ S \to T \circ G$ be the associated natural isomorphisms. Then there exists an equivalence $(\hat{F}, \hat{G}, \eta, \epsilon)$ between $\text{Coalg}(T)$ and $\text{Coalg}(S)$. On objects, we have

\[
\hat{F}(f : a \to T(a)) = k_a \circ F(f)
\]

\[
\hat{G}(g : b \to S(b)) = l_b \circ G(g)
\]

The action of $\hat{F}$ and $\hat{G}$ on arrows is just the same of $F$ and $G$, respectively.

**Proof.** We have to show that $\eta$ is a natural isomorphism from $\text{Id}_{\text{Coalg}(T)}$ to $\hat{G} \circ \hat{F}$, i.e., that it is a coalgebra isomorphism for each coalgebra $f : a \to T(a)$, and that it commutes with arrows. The first part is shown by commutativity of the diagram below:

\[
\begin{array}{ccc}
T(a) & \xrightarrow{\eta_T(a)} & G(F(T(a))) \\
\downarrow^{G(k_a)} & & \downarrow^{G(k_{T(a)})} \\
T(G(F(a))) & \xrightarrow{\eta_T(G(F(a)))} & G(T(G(F(a))))
\end{array}
\]

The upper square commutes by naturality of $\eta$. For the lower triangle, consider the following composition of natural transformations:

\[
\begin{array}{ccc}
T(a) & \xrightarrow{\eta_T(a)} & G(F(T(a))) \\
\downarrow^{T(\eta_a)} & & \downarrow^{G(k_{T(a)})} \\
T(G(F(a))) & \xrightarrow{\eta_T(G(F(a)))} & G(T(G(F(a))))
\end{array}
\]

\[
\begin{array}{ccc}
T(G(F(a))) & \xrightarrow{G(\eta_G(F(a)))} & G(T(G(F(a)))) \\
\downarrow^{G(k_{G(F(a))})} & & \downarrow^{G(k_{T(G(F(a))})} \\
G(S(F(a))) & \xrightarrow{G(\eta_T(G(F(a)))} & G(T(G(F(a))))
\end{array}
\]
Commutativity of the outer square in the diagram above is ensured by com- positionality of natural transformations. To see that it is the same as commutativity of the lower triangle in the previous diagram, one should look at how \( l \) is defined in terms of \( k \). By lemma 92, \( l^{-1}_F(a) = G(S(\epsilon_{F(a)})) \circ G(k_G(F(a))) \circ \eta_{T(G,F(a))} \). Since any equivalence of categories is also an adjunction, where \( \eta \) is the unit and \( \epsilon \) the counit, we have \( \epsilon_{F(a)} \circ F(\eta_{a}) = id_a \), and since \( F(\eta_{a}) \) is an isomorphism with inverse \( F(\eta_{a}^{-1}) \), we have \( \epsilon_{F(a)} = F(\eta_{a}^{-1}) \). Replacing it in the equation above, we obtain \( l^{-1}_F(a) = G(S(F(\eta_{a}^{-1}))) \circ G(k_G(F(a))) \circ \eta_{T(G,F(a))} \). It is now easy to see that this makes commutativity of the lower triangle in the first diagram and of the outer rectangle in the second diagram equivalent conditions. This concludes the proof of commutativity of the first diagram we presented. Commutativity with coalgebra morphisms is easy since \( \eta \) is a natural isomorphism (the difficult part of the proof is to show that it lifts to a coalgebra isomorphism).

The above theorem constructively derives, from a fully abstract semantics in \( \textbf{Coalg}(T) \) of a given calculus, a fully abstract semantics in \( \textbf{Coalg}(S) \). Thus, the HD-automata semantics of various nominal calculi can be obtained from well known results in presheaf categories. Then, the definition of the functors we gave in this thesis can be used to obtain an implementable definition of such semantics.

**Theorem 94.** Product, coproduct, countable power set and name abstraction in \( \textbf{FSAlg}^\pi \) are equivalent to the corresponding functors in \( \textbf{NSet} \).

**Proof.** For the product, which is a limit, just observe that categorical equivalence preserves limits, and that these are unique up-to isomorphism. Hence, the definition of product obtained by equivalence must be isomorphic to the one that we gave, and proved to be the limit, in \( \textbf{NSet} \). For the coproduct, the same argument holds. Equivalence between the two notions of name abstraction is theorem 77. Equivalence between the two notions of power set is theorem 88.

We can now state our final coalgebra theorem in \( \textbf{NSet} \), into which we also include subfunctors, by lemma 23 and the fact that equivalences preserve monos.

**Theorem 95.** Functors obtained by composition of product, coproduct, countable power set and name abstraction, and their subfunctors, have a final coalgebra in \( \textbf{NSet} \).
Chapter 8

Examples

8.1 Example: Causal Automata as Coalgebras

In [MP97], causal automata, that were a starting point to define HD-automata, were introduced to represent the history-preserving semantics of place-transition Petri nets. In this setting, names have a completely different meaning than in the π-calculus: they represent events that are generated each time a transition is fired, and can be subsequently referenced to denote a causal dependency. Names in each state are local, and bisimilarity is a ternary relation featuring pairs of states, and name mappings between them.

The definition of a causal automaton follows.

Definition 96 (Causal automaton). Given a set Act of labels, a causal automaton is a tuple \( (Q, w, \rightarrow, q^0) \) where

- \( Q \) is a set of states
- \( w : Q \rightarrow \mathcal{P}_{\text{fin}}(\omega) \) associates to each state a finite set of names.
- \( q^0 \) is the initial state, with \( w(q^0) = \emptyset \)
- \( \rightarrow \) is a (finite) set of transitions of the form \( q \xrightarrow{a,D} q' \), where
  - \( q, q' \in Q \) are the source and target states
  - \( a \in \text{Act} \) is the label
  - \( D \subseteq w(q) \) are the dependencies of the transition
\[ \sigma : w(q') \xrightarrow{\text{inj}} w(q) + 1 \] is an injective renaming for the transition, where the additional name \( * \in 1 \] represents a freshly generated name that denotes the current transition in the target state.

These automata-like structures are equipped with a notion of causal bisimulation.

**Definition 97 (causal bisimulation).** Given two causal automata \( \langle Q_N, w_N, \rightarrow, q_0^N \rangle \) and \( \langle Q_M, w_M, \rightarrow, q_0^M \rangle \), a set of triples \( R \) is a causal bisimulation if:

- whenever \( \langle q_N, d, q_M \rangle \in R \), then \( q_N \in Q_N, q_M \in Q_M \) and \( d \) is a partial bijection between \( w_N(q_N) \) and \( w_M(q_M) \).
- \( \langle q_0^N, \emptyset, q_0^M \rangle \in R \)
- whenever \( \langle q_N, d, q_M \rangle \in R \) and \( q_M \xrightarrow{a,D} \sigma_M q'_M \), then there exists some transition \( q_N \xrightarrow{a,d(D)} \sigma_N q'_N \) and some \( d' \) such that \( \langle q'_N, d', q'_M \rangle \in R \), and \( d'(n) = m \) implies \( \sigma_N(n) = \sigma_M(m) = * \) or \( d(\sigma_N(n)) = \sigma_M(m) \)
- whenever \( \langle q_N, d, q_M \rangle \in R \) and \( q_N \xrightarrow{a,d^{-1}(D)} \sigma_N q'_N \) and some \( d' \) such that \( \langle q'_N, d', q'_M \rangle \in R \), and \( d'(n) = m \) implies \( \sigma_N(n) = \sigma_M(m) = * \) or \( d(\sigma_N(n)) = \sigma_M(m) \)

Causal automata and their bisimulation were used in [MP97] to give a fully abstract model of the history-preserving bisimulation for place-transition Petri nets that was defined in [BDKP91]. The following quote introduces causal bisimulation in [MP97]: “On causal automata a bisimulation cannot simply be a relation on states: also a correspondence between the names of the states has to be specified and the same pairs of states can be in relation via more than one correspondence. This correspondence is partial in general, since we do not want to oblige two equivalent states to have the same number of private names”. The categorical product of named sets is apt to capture such a notion. Notice how in the case of Petri nets, it is more natural than in the \( \pi \)-calculus to use local names, since these are the names of locally generated events.

Causal automata can be viewed as coalgebras in \( \text{NSet} \) employing the functor

\[ C(N) = \mathcal{P}_{\text{fin}}(L \times H(N)) \]

The named set \( L = \langle Q_L, S_L \rangle \) has, as underlying set, the product \( \text{Act} \times \mathcal{P}_{\text{fin}}(\omega) \). Each element of this product represents the pair \( a, D \) that labels a transition. The symmetry
8.1. EXAMPLE: CAUSAL AUTOMATA AS COALGEBRAS

is defined as $S_L((a, D)) = \{id_D\}$, that is, each label has no symmetry, and its names are those in $D$. $L$ is isomorphic to the named set $\text{Act} \times \mathcal{P}_{\text{fin}}(\omega)$ where $\text{Act}$ is seen as a named set without names, and $\mathcal{P}_{\text{fin}}(\omega)$ is seen as a named set having symmetry $S(p) = \{id_p\}$.

In this section we will not use symmetries, so $\text{Symset}$ restricts to its full subcategory $I$. For readability, we can then employ injective functions rather than singletons of injective functions to represent arrows of $\text{Symset}$.

**Definition 98** (From causal automata to $C$-coalgebras). Given a causal automaton $\langle Q_N, w_N, \longrightarrow, q_0^N \rangle$, we define the corresponding coalgebra $tr : N \to C(N)$ as follows.

The named set $N$ of states is defined as $N = \langle Q_N, S_N \rangle$, where $S_N(q) = \{id_{w(q)}\}$, that is, the symmetry of each state is trivial, and the support is given by the function $w$. For each $q \in Q_N$, consider the set of its outgoing transitions:

$$\{\langle l_i, q_i', \sigma_i \rangle \mid q \xrightarrow{a, D} \sigma_i q_i' \land l_i = \langle a, D \rangle\}$$

For each $i$, let $\text{in}_i$ denote the inclusion of names of $l_i$ into names of $q$, $\sigma' = (\sigma_i)_{w(q_i) \sigma^{-1}(\ast)}$ the mapping of names of $q_i$ into names of $q$. Let $\text{in}_i^1$ and $\text{in}_i^2$ denote $\text{MCL}(\langle \text{in}, \sigma_i \rangle)$ in $\text{Symset}$, and let $t_i$ denote the unique mediating arrow. Finally, let $\langle \text{in}_i \rangle$ denote the multi-colimit of all the $\langle t_i \rangle$ in $\text{Symset}$, and $j$ the obtained unique arrow. Then we pose

$$h(q) = \{\langle \langle l_i, q_i, \sigma^{-1}(\ast) \rangle, \text{in}_i^1, \text{in}_i^2, \text{in}_i \rangle\}$$

$$\Sigma(q) = \{j\}$$

The definition is obtained as follows: for each transition of the causal automaton, first of all we identify the pair $\langle q_i, \sigma^{-1}(\ast) \rangle$, that is, the element of $H(N)$ that abstracts name $\sigma^{-1}(\ast)$ in the destination state $q_i$. Then, we observe that names of the label $l_i$ and of the state $q_i$ are related by their image into names of the source state $q$, and in particular those of $l_i$ are just included in those of $q$, while the names of $q_i$, except $\sigma^{-1}(\ast)$, are mapped to names of $q$ via $\sigma_i$. Thus, we have two arrows, namely $\text{in}$ and $\sigma'$, with a common target. These two arrows are represented by their multi-coproduc
t, thus making the quadruple $\langle l_i, \langle q_i, \sigma^{-1}(\ast) \rangle, \text{in}_i^1, \text{in}_i^2 \rangle$ an element of $L \times H(N)$. The unique arrow $t_i$ takes back the names of this element to the names of the source state $q$. Then, the whole (finite) set of transitions is made into an element of the finite power set by the multi-coproduc
t on all the $t_i$, whose unique mapping $j$ still takes names of the whole set of transitions into names of the source state $q$. 
A bisimulation between two $\mathbf{C}$-coalgebras is now a subset of the product $N \times M$. Since there are no symmetries, the name mappings that come from the multi-coproduct in definition 80 are just cospans of injective functions, that cover their target, and that are quotiented by isomorphism of cospans. Indeed, each so defined cospan is a partial bijective mapping of names as required by the definition of causal bisimulation. Given two $\mathbf{C}$-coalgebras $tr_1 : N \to \mathbf{C}(N)$ and $tr_2 : N \to \mathbf{C}(M)$, a bisimulation between $tr_1$ and $tr_2$ is thus a subset of $Q_{N \times M}$ that extends to a coalgebra $b : R \to \mathbf{C}(R)$. Commutativity of the associated diagram coincides with the definition of causal bisimulation, if $R$ respect the additional constraint $\langle q^0_N, q^0_M, id_{\emptyset}, id_{\emptyset} \rangle \in R$.

8.2 The $\pi$-calculus

In this section we get back to the early semantics of the $\pi$-calculus, to show how all the results we have presented can be put together to obtain, from a semantics expressed by the means of nominal sets, one that uses local names.

We employ the simplified functor of section 4.2

$$T(X) = \mathcal{P}_\omega(\mathcal{L} \times \delta(X))$$

where the permutation algebra of labels $\mathcal{L}$ has the carrier

$$\{tau, in(x, y), out(x, y), bout(x) \mid x, y \in \omega\}$$

equipped with the obvious permutation action. The functor is included in our framework, hence it has a final coalgebra. As we already discussed, we can give a fully abstract semantics of the $\pi$-calculus using the final coalgebra of this functor. To clarify how the semantics is defined, we present it using the new functor in Fig. 8.1. Notice the usage of the embedding $old$ to represent transitions that do not allocate fresh names.

Observe that the semantics as it is defined is not “implementable” (e.g., it is not possible to evaluate iteration along the terminal sequence even in the finite state case) even in $\mathbf{NSet}$, because the input action gives rise to infinite transitions that one has to represent.

A technique which is often used to tackle this problem is to define a finitary transition system, based on the early semantics, that contains the finite number of “meaningful” input transitions, i.e. those that receive a non-fresh name, plus a single transition that represents the infinite ones that receive a fresh name. A very good argument to do so is
8.2. THE $\pi$-CALCULUS

that all the transitions carrying a fresh name give rise to bisimilar states, up to injective relabelling. However, the problem of redundant names makes this approach incorrect. Consider for example the following two agents:

$$P(x,y) = x(z).\bar{z}z.0 || (\nu w) w y.0$$

$$Q(x) = x(z).\bar{z}z.0$$

Observe that the prefix $\bar{w}y$ in the definition of $P$ does not trigger any output transition, because it is immediately under the scope of a restriction over its subject. Thus, the two agents are bisimilar. However, if we base our choice of “interesting” input transitions only on free names of a process, we get two finitary systems that are not bisimilar, since $P(x,y)$ gains a spurious free input transition labelled with $xy$ which is not present in $Q(x)$: name $y$ is free in $P(x,y)$, but it is not observable in the semantics, i.e. it is redundant. Free names of a process are a syntactic notion. Instead, we are interested in observable names, belonging to the realm of the semantics. Unfortunately, redundancy of names is not decidable in general. Figures 8.2 and 8.3 depict the above example and the incorrect approximation giving rise to non-bisimilar systems.

In [FMT05a] a minimisation algorithm for finite-state $\pi$-calculus agents was given. We first provide an intuition on the method by which the above problem is tackled there. Then, we show that the idea can be formalised using a subfunctor of an accessible endofunctor that contains the semantics of the $\pi$-calculus.

The key idea is to change the action of the functor on arrows from that of the power set to a normalising variant of it. The action of this modified functor removes from the destination $T(f)(p)$ of a coalgebra $f$ applied to a process $p$ all the free input transitions that are proved to receive a fresh name in one step, that is, it removes all the pairs $\langle \text{in}(x,y), \text{old}(q) \rangle \in T(f)(p)$ such that $\langle \text{bin}(x), \text{hid}^y(q) \rangle \in T(f)(p)$. Notice that the two
\[ P(x, y) \sim Q(x) \]

Figure 8.2: standard LTS

Figure 8.3: The incorrect approximation
destination states must be exactly the same, not just bisimilar, which corresponds to say that the free input transition is proved to be redundant in exactly one step.

A correct semantics is then obtained starting from the incorrect approximation that we mentioned above, by iteration along the terminal sequence: at each step of the minimisation procedure, we have an \( n \)-step approximation of the image of an agent in the final coalgebra, where all input transitions that can be proved redundant in \( n \) steps have been removed. When the fixed point is reached, no redundant transitions are present.

The definition of the functor in [FMT05a] was given directly on named sets, and its complexity was not factored out using subfunctors. Even though termination in the finite state case was proved, the functor employed there is not tailored to represent the semantics of the \( \pi \)-calculus in the infinite-state case. We will show that the representation technique employed there can be extended to the full semantics of the \( \pi \)-calculus by giving a final coalgebra semantics where the set of transitions in each step is finite.

Instead of proving directly the correctness of a functor in \( \textbf{N} \text{Set} \), we adopt the same technique used in [FMT05a] to define a normalising functor in \( \textbf{FSAlg}^\pi \), of which we formally prove correctness, and existence of the final coalgebra. Then results of §7 ensure existence of a corresponding functor in \( \textbf{N} \text{Set} \), which is very close to that of [FMT05a].

In the following we define a functor \( P : \textbf{FSAlg}^\pi \rightarrow \textbf{FSAlg}^\pi \) such that for each permutation algebra \( A \) the resulting algebra \( P(A) \) only contains finite subsets. \( P \) is not included “as is” in our framework of accessible endofunctors, because it employs normalisation on arrows, thus we still have to prove that it has a final coalgebra. We show that \( P \) is isomorphic to a subfunctor \( P' \) of \( P_{\omega}(\mathcal{L} \times \delta(-)) \), that by definition only contains \( \pi \)-calculus transitions. This formally justifies the normalisation step as a correct representation of a “well behaved” subfunctor of the countable power set. By lemma 23, \( P' \) admits a final coalgebra, so also \( P \) does.

In the following, for \( A = \langle A, \{\pi_A\} \rangle \) permutation algebra, let \( \sigma^{(z,y)} \) denote the permutation swapping \( z \) and \( y \), and \( Ein_A \) denote the set of all sets of the form

\[
\{\langle \text{inp}(x,y), \sigma^{(z,y)}_{\delta A}(\text{old}_A(a)) \rangle \mid y \in \omega \setminus \{x\} \setminus \text{supp}_A(a)\}
\]

for a fixed \( x \in \omega \), an element \( a \) of \( A \), and a name \( z \in \text{supp}_{\delta A}(\text{old}_A(a)) \). This is only used in the definition of \( P' \), to make it contain all early input transitions, without having to resort to an (undefined, in a permutation algebra) notion of non-injective substitution.
Definition 99 (functor $P'$). The functor acts on objects as

$$P'(A) = \{ p \cup (\bigcup p') \mid p \in P_{fin}(\mathcal{L} \times \delta(A)) \land p' \in P_{fin}(Ein_A) \}$$

and it acts on arrows in the same way as $P_{\omega}(\mathcal{L} \times \delta(-))$.

In the definition, each $p'$ is a finite subset of $Ein_A$, hence a set of sets. Each element of $p'$ is a set of early input transitions that are obtained from each other by injective substitution. $P'$ is obviously a subfunctor of $P_{\omega}(\mathcal{L} \times \delta(-))$ by inclusion of sets, and it contains the semantics of the $\pi$-calculus of Fig. 8.1.

Now we want to define the functor $P$. Since bound input transitions are not present in the semantics of the $\pi$-calculus, we have to define an additional set of labels $L^b$ whose carrier $L^b$ is equal to $L \cup \{ binp(x) \mid x \in \omega \}$, where $L$ is the carrier of $\mathcal{L}$. The latter only features the subject of the transition, since the received name is considered fresh.

For $A$ permutation algebra and $p \in P_{fin}(L^b \times \delta(A))$, consider the function

$$norm_A(p) = p\{ \langle inp(x,y),old_A(a) \rangle \mid \langle binp(x),hid_Y^a(a) \rangle \in p \}$$

This removes from $p$ all the free input transitions $inp(x,y)$ such that there is a bound input transition on $x$, going into the same state, with $y$ replaced by a fresh name.

Definition 100 (functor $P$). The functor $P$ acts on objects as

$$P(A) = \langle P, \left\{ \pi_{P_{fin}(L^b \times \delta(A))} \right\} \rangle$$

where

$$P = \{ norm_A(p) \mid p \in P_{fin}(L^b \times \delta(A)) \land \neg \exists a \in A, x \in \omega. \langle binp(x), old_A(a) \rangle \in p \}$$

and on arrows as

$$P(f : A \rightarrow B) = norm_B \circ P_{fin}(L^b \times \delta(f))$$

The condition on bound input transitions is there because we are only interested in bound input transitions that actually allocate a fresh name. Notice that the definition in practice identifies a subfunctor of $\delta$ that only contains elements with hidden names. The isomorphism between the two functors is established by the following theorem.

Theorem 101. The functors $P$ and $P'$ are isomorphic.
8.2. THE $\pi$-CALCULUS

Proof. A natural transformation from $P$ to $P'$ is given by

$$\iota_A(p) = \{ \langle l, a \rangle \in p \mid l \neq \text{binp} (x) \} \cup \bigcup_{(\text{binp}(x), \text{hid}_{\iota_A}(a)) \in p} \{ \langle \text{inp}(x, y), \sigma_{\delta(A)}(\text{old}_A(a)) \rangle \mid y \in \omega \setminus \{x\} \setminus \text{supp}_A(a) \}$$

To see that $\iota_A$ is injective, consider $p, q \in P(A)$. If $p \neq q$, then there is a transition $t$ such that either $t \in P$ and $t \notin q$, or $t \in q$ and $t \notin p$. If $t$ is not an (either free or bound) input, $i_A(p) \neq i_A(q)$ by definition of $\iota_A$. If $t$ is a free input transition, it may be possible in principle that a bound input transition $t'$ such that $t' \in p$ and $t' \notin q$ is turned into a set of free input transitions containing $t$. However, this possibility is ruled out by $p$ and $q$ being obtained by the function $\text{norm}$ (by definition of $P$ on objects). Observe that this is the reason why we obtain an isomorphism, and the difference with the incorrect approximation that we explained above. If $t$ is a bound input, then it gives rise to an infinite set of transitions that are present in one of $\iota_A(p)$ or $\iota_A(q)$ but not in the other one.

The inverse $\iota^{-1}_A(p')$ is obtained by looking at the definition of $P'$: we have $p' = p_1 \cup \bigcup_{i \in I} p_i$ where $p_1$ is a finite set of transitions, $I$ is a finite set that indexes the set of sets $p_i$, and each $p_i$ is an infinite set of input transitions obtained from each other by swapping two names. Each $p_i$ is replaced in $\iota^{-1}_A$ by a corresponding bound input transition in the obvious way.

For commutativity with arrows, consider an arrow $f : A \to B$ and an element $p$ of $P(A)$. If there are no bound input transitions in $p$, or if $\text{norm} (P(f)(p)) = P(f)(p)$, commutativity is trivial. If there is at least a bound output transition in $p$ and $\text{norm} (P(f)(p)) \neq P(f)(p)$, then we have at least a free input transition $t = \langle \text{inp} (x, y), \text{old}_A(a) \rangle \in p$, and a bound input transition $t' = \langle \text{binp} (x), \text{hid}_A(b) \rangle \in p$, such that $f(a) = f(b)$. Thus, we have

$$\{ \langle \text{inp}(x, y), \sigma_{\delta(A)}(\text{old}_A(b)) \rangle \mid y \in \omega \setminus \{x\} \setminus \text{supp}_A(a) \} \subseteq i_A(p)$$

and, by taking in account $f(a) = f(b)$,

$$t'' = \{ \langle \text{inp}(x, y), \sigma_{\delta(B)}(\text{old}_B(f(a))) \rangle \mid y \in \omega \setminus \{x\} \setminus \text{supp}_B(f(a)) \} \subseteq P'(f)(i_A(p))$$

On the other hand, $t'$ is sent by $P(f)$ into $\langle \text{binp} (x), \text{hid}_B(f(b)) \rangle = \langle \text{binp} (x), \text{hid}_B(f(a)) \rangle$, and $\iota_B$ turns this bound transition into exactly $t''$. Observe that $t''$ also includes the transition obtained from the free input transition $t$ above via $P'(f) \circ \iota_A$, thus the bound input transition in $P(f)(p)$ suffices to represent both of them. \qed
By theorem 23, \( P' \) (thus, \( P \)) is accessible, so it has a final coalgebra. Then, by theorem 91, the following corollary holds.

**Corollary 102.** The functor corresponding to \( P \) in \( \text{NSet} \) has a final coalgebra.

By the above result, the semantics of the \( \pi \)-calculus is representable using finitary transition systems in \( \text{FSAlg}^\pi \), that can be seen as labelled transition systems plus name binding along transitions. Such a representation solves the problem, which is inherent to the early semantics, that early input transitions can be infinitary. The HD-automata definition then introduces garbage collection. We now examine how transitions are concretely described in the HD-automaton for the \( \pi \)-calculus.

For each named set \( N \), the named set of transitions is a subset of \( \mathcal{P}_{\text{fin}}(\mathcal{L}^b \times \delta(N)) \). Elements of the underlying set \( Q \) are thus sets of pairs \( \langle q_i, i_n \rangle \) coming from the power set. Each \( q_i \) in turn is an element of the product \( \mathcal{L} \times \delta(N) \). Thus, each \( q_i \) is either in the form \( \langle l, q, in_1, in_2 \rangle \), or in the form \( \langle l, \langle q, n \rangle, in_1, in_2 \rangle \), where \( l \in L^b \) (as a canonical representative of its orbit), \( q \in Q_N \), \( n \in \|q\|_N \). The former is a transition without bound names, while the latter is a transition having a bound name \( n \).

This representation of transitions is very similar to the one that was employed in [FMT05a]. Here we will keep the discussion at an intuitive level. A full proof of isomorphism of the two functors would be just a technical matter, and is completely unnecessary: the functor of [FMT05a] has been proved correct with respect to its purpose, which is to minimise finite state systems, and the functor \( P \) that we define is a correct representation in \( \text{NSet} \) of the final coalgebra semantics of the \( \pi \)-calculus in \( \text{FSAlg}^\pi \). Any implementation of iteration along the terminal sequence for \( P \) must return the same result of [FMT05a], that is, minimisation of finite state systems.

In [FMT05a], the functor is defined using the so-called quadruples, consisting of a label, a state and two name mappings from names of the label and of the state into a set of common names. This common set is augmented with an additional element that serves the purpose to mark an eventual name of \( q \) which is mapped into it as fresh (much in the same way as this is done in causal automata). The union of many quadruples with the same target set of names, corresponding to an element of the finite power set, is called a bundle. There, a fresh name can be generated in each transition (which in \( P \) is obtained using \( \delta \)), and the normalisation step is defined in the same way as the function \( \text{norm} \). Therefore, the techniques used in [FMT05a] actually give a garbage collecting, finitary (in the sense of the transition system) implementation of the final coalgebra
semantics of the \( \pi \)-calculus in the finite state case.

A final semantics for the \( \pi \)-calculus in the category of named sets, extending the finite state case, allows, among other things, the implementation of on-the-fly methods for infinite state systems.

**Remark 103.** A different way to tackle the problem of approximating the semantics could be to add redundant transitions to each agent to match the free names of other agents against which one would like to check bisimilarity. This would also work for partition refinement, that could be carried out by artificially enlarging the set of “free” names of each state to the union of the names of all the states of the system. Even though this would lead to a correct partition refinement procedure, differently from the technique we present here employing normalisation, it would not remove redundant transitions and redundant names. Not removing redundant transitions introduces (at least) two problems. First, one cannot know "a priori" for each state how many transitions need to be added. The union of all names of all reachable states should be computed in advance. Thus, one cannot store the minimal model \( m \) for subsequent usage, because, e.g. when checking bisimilarity of a new model \( m' \) against it, \( m \) should be recomputed according to the free names of \( m' \). Second, such a model can become exponentially less efficient than the one obtained by normalisation, since redundant transitions are not eliminated. For example, each redundant transition multiplies the number of paths that a model checker would need to examine. This would entirely defeat the usefulness of minimising a system before static analysis.
Part IV

Conclusions and Future Developments
Chapter 9

Future Work

We have presented a theory of accessible functors to define HD-automata in a compositive

dional way. Starting from the theory of permutation algebras, we first have provided the
ecessary theory to take in account garbage collection of unused names, by defining a
uitable abstraction functor in the category of named sets. This is essential for the defi-
dition of algorithms to handle the syntax and the semantics of name passing calculi. We
have also shown that, using the algebraic definition of nominal sets, a very simple theory
orphism induces a number of commonly used operations in the category of nominal
ets: name abstraction, the freshness relation, and the concretion, or reveal, operation.
Then, we have shown how locality of names affects the definition of the categorical pro-
don and the power set in a significative way, and how the symmetry of a set of elements
can be greater with respect to that induced from single components. Finally, we have
been able to give a coalgebraic characterisation of history dependent bisimulation and of
the normalisation of the early semantics of the π-calculus of [FMT05a]. Ongoing work
is in many directions.

Permutation groups and polynomial computation

In this thesis we have mainly given a modular tractation of finite-state methods to rep-esent the semantics of nominal calculi. This gives one the possibility to implement, for
example, algorithms that operate on the whole state space of a system. The immediate
ext question is how efficient this process can be made.

We already have a space-efficient representation of minimal systems in [FMT05a],
that employs the generators of a group to represent the symmetry of each state. Gen-

operators are logarithmic in the size of the group they represent. However, in the mihda toolkit, this advantage is not reflected in the execution time of algorithms, that unfold the whole set of symmetries in many cases.

In particular, it is of interest to define dictionaries that contain elements of the categorical product of two named sets. As of the definition of the categorical product, when a quadruple \( \langle q_1, q_2, in_1, in_2 \rangle \) is in the dictionary, we need an efficient method to determine if a new quadruple \( \langle q'_1, q'_2, in'_1, in'_2 \rangle \) is represented or not by the first one, that is, if \( MCL((in'_1, in'_2)) = (in_1, in_2) \).

Preliminary research aimed at model checking showed that the problem can be reduced to check whether, in a permutation group over a finite domain \( D \), there is a permutation \( \sigma \) such that, for all the names \( n_1, \ldots, n_k \) in \( S \subseteq D \), we have \( \sigma(n_1) = n'_1, \ldots, \sigma(n_k) = n'_k \), for \( n'_1, \ldots, n'_k \) fixed. This is one of the problems tackled in the works by Eugene M. Luks (see [Luk93] for an overview). In particular, this problem can be solved in polynomial time using the generators. This is promising as a starting point to develop more efficient minimisation and model-checking algorithms exploiting the symmetry reduction provided by named sets.

Model checking

As mentioned in §1, the work presented here stems from preliminary research on finite-state verification methods for nominal calculi. In particular, it is expected that a model checking algorithm can be developed, that operates directly on minimal models. As minimal models are canonical representatives of bisimilar systems, the chosen logic has to characterise bisimulation. Sources of inspiration can thus be the Hennessy-Milner logic of [MPW93], and the nominal \( \mu \)-calculus of [Dam93]. The logic developed in [BK07] is also closely related, since it is defined using presheaf categories and features a permutation operation on formulas. The definition of the basic functors in this work should leverage the complexity of defining an interpretation of logic formulas directly on HD-automata. In cases with symmetry, it is easily seen that the minimal model can be exponentially smaller than the unfolded semantics. This is of particular interest with calculi having a parallel composition operator: putting \( n \) bisimilar processes in parallel typically results in a super-polynomial (with respect to \( n \)) explosion of the number of states that are symmetrical in the choice of which of the \( n \) processes performs a transition at each step. It is easy to find useful cases (such as simulating the join of \( n \) computations
in the π-calculus) where the minimal model has a number of states that does not grow
with n because symmetry of the start state represents the whole system. By exploiting
polynomial time algorithms for permutation groups, model checking in this case could
benefit of an exponential reduction of its complexity in turn. Indeed, it has to be
checked whether the minimisation procedure is intrinsically exponential or not. Even if it
is, however, there are good reasons to pursue this approach. On one hand we have an
optimisation if we have to check more than one formula on the model, since minimisation
only happens once. On the other hand, it should be investigated if it is possible to derive
an approximated minimisation procedure that is not exponential in the average case. We
remark that, even though there is great interest in symmetry-based model checking (see
e.g. [SGE00]), symmetries in existing systems are typically found "by hand", and not
inferred automatically as it can be done with minimisation of HD-automata.

On-the-fly methods

The minimisation procedure developed in [FMT05a] only works in the finite state case,
and is not performed on-the-fly, that is, the whole state space of the starting system has
to be built before minimisation. There are obvious reasons that may make it impractical,
in particular a reduction strategy in model checking is often required quite because there
is not enough space to build the whole state set. Hence, an on-the-fly minimisation
procedure is of interest, that should compute refinement steps on demand, driven by the
model checking algorithm. This also enables one to verify infinite state systems, even
though the model checking problem is no longer decidable in this case.

Spatial logics for nominal calculi

Spatial logics for nominal calculi are interesting to reason about properties related to
the presence of hidden names in states (for example, secrecy properties in formalisms for
secure communication). Works [CC03] and [Cai04] are among the standard references on
the topic. The approach of spatial logics for nominal calculi was pursued in [VC03], where
a simplified version of HD-automata was used to build a spatial logic model checker that
is able to verify secrecy properties of cryptographic protocols. The functors we define
in this work should be of help in defining observational models of spatial logics, along
the lines of [Mon04], thus making it easier to develop similar analysis tools. Also, the
relationship between the counit of the adjunction between abstraction and concretion
that we presented in this thesis, and spatial operators over names, should be studied in detail.

Quantitative reasoning

In [CF07], the author and Gian Luigi Ferrari provide a link between spatial decomposition operators and quantitative analysis of systems as done in [LM05]. $C$-semirings are used in place of boolean logics to define a notion of quantitative satisfaction of logic formulas. Remarkably, quantities are not defined in system states, but rather they are inferred from the number of available distinct subsystems that satisfy the given formula. Spatial decomposition operators under the scope of a name restriction are difficult to tackle. For example, the logic in [Cai04] characterises systems up-to structural congruence, and not to an appropriate (concurrent) bisimulation. Presheaf categories or named sets could prove the right tool to define an appropriate, semantic notion of satisfaction for parallel decomposition operators over nominal calculi, whose model checking algorithms then would be defined with the help of the functors we give in this thesis.

Stone duality

As we mentioned several times, there is great interest in defining appropriate logics for reasoning about systems whose semantics is expressed using HD-automata. The most important point to be able to exploit minimisation results to verify formulas of logics that characterise systems up-to bisimulation. A systematic way to derive such logical characterisations is given by the lifting of Stone duality to categories of coalgebras that was given in [BK05] and subsequent works. In particular [BK07] applies the framework to the $\pi$-calculus using presheaf categories. Defining the logic directly on HD-automata is of interest because then also formulas have local names, thus the product of the state space of the system and the subformulas of the formula to be checked against it can be greatly reduced. Further research is required, but having defined a basic set of accessible functors might allow us to exploit a modular tractation of Stone duality for coalgebras. Related works are [BK06] and the modular presentation of functors that was first presented in [BK07] and then in [KP08].
Service-oriented computing

Applications of verification up-to bisimulation are important for service-oriented computing, since services are seen as black-boxes that we can only interact with, hence identified by observational equivalence.

In [CFPT08], the author, Gian Luigi Ferrari, Marco Pistore and Emilio Tuosto use HD-automata as a syntax-independent formalism to check compatibility of services at binding time in service-oriented computing. Service requests are modelled as pairs of $\langle C_o, C_r \rangle$ where $C_r$ describes the (abstract) behaviour of the searched service and $C_o$ the (abstract) behaviour guaranteed by the invoker. Symmetrically, service publication consists of a pair of $\langle S_o, S_r \rangle$ such that $S_o$ provides an (abstraction of) of the behaviour guaranteed by the service and $S_r$ yields the requirement imposed to invokers. An invocation $\langle C_o, C_r \rangle$ matches a published interface $\langle S_o, S_r \rangle$ when $C_o$ simulates $S_r$ and $S_o$ simulates $C_r$.

This idea is very basic, but the presence of names in HD-automata make it appealing to specify service binding where generation of fresh resources is possible, in particular session based protocols. The idea of using simulation is that the invoked service can provide more possible usage patterns than those requested by the client, e.g. because it provides different versions of the same protocol. The definitions given there are more or less those from [FMT05a]. Employing the functors defined in this thesis should lead to a cleaner and more general formulation. Then, simulation could be defined in a purely coalgebraic way. An extension of the framework to game-theoretical models, to accommodate the alternation of external and internal choice between two parties in a session, is of great interest here.

There is also ongoing work on the side of specification languages for service-oriented computing. In [CFGS08a] the author, together with Gian Luigi Ferrari, Roberto Guanciale and Daniele Strollo, defined an LTS semantics for the Signal Calculus, an asynchronous process calculus featuring multicast communication. The calculus relies on explicit modelling of the communication structure of the network (communication flows), and on handling sessions, even multi-party. Its definition is strongly motivated by the practical needs of Service-Oriented Computing, and there exists a Java implementation, called JSCL, with a graphical modelling framework.

The LTS semantics was defined to the aim of introducing formal verification techniques, and introduced the usual side conditions on freshness of names. It is expected
that the verification methods already employed for the $\pi$-calculus can be applied also to SC/JSCL, equipping the graphical modelling framework with a bisimulation checker and a model checker, exploiting this thesis work and the perspective model checking algorithm.

The work on the SC framework has been continued by the same authors in [CFGS08b], by equipping the calculus with a more high-level policy specification language that takes a global point of view on the coordination of services. Two different approaches can be adopted to tackle service coordination: orchestration and choreography. In that work, we have introduced a formal methodology purposed to handle coordination among services from the perspective of a global observer, in the spirit of choreography models. In particular, there we address the problem of verifying compliance and consistency between the design of service interactions and the choreography constraints. The definition of compliance is given in terms of an embedding of SC into the policy calculus NCP, and a weak bisimulation. Indeed, NCP is a nominal calculus, and to actually provide an implementation of the compliance checking we foresee to use HD-automata.

Generalising named sets to categories with local interfaces

Named sets on one side, and permutation algebras/nominal sets/the Schanuel topos on the other side, model in a neat way calculi based on injective relabelings. For calculi with fusions of names, or the open semantics of the $\pi$-calculus, things are less clear. Presheaf models whose index category is different from $1$ have been successfully used. It should be possible to redo the same construction that obtains a permutation algebra from a sheaf in $\text{Set}^1$, and then a named set from this permutation algebra, on these different categories of (pre)sheaves. As an example, the HD-automata with distinctions of [FMT+05b] might be included in the general framework. The general construction should take advantage of the idea found in Sam Staton’s thesis to represent named sets as free coproduct completions. As we mentioned at the end of section 6.4, of particular interest is the categorical product of obtained categories. The relationship between presheaves and free coproduct completions has to be examined and used in this case.

Many questions arise: what is the notion that corresponds to symmetry, when operations are no longer permutations? Is there still an efficiency advantage as it happens in the case of permutation groups? Exploiting the standard fact that any group may be represented as a permutation group, is it possible to represent any category of gen-
eralised named sets as ordinary named sets? What would be gained or lost in efficiency doing so?

We mention two possible case studies. In [BBCG08], the author, together with Filippo Bonchi, Marzia Buscemi and Fabio Gadducci studied a presheaf model for the calculus of explicit fusions, a calculus featuring symmetric binding. Calculi with symmetric binding, in the spirit of the fusion calculus, pose new challenges. In the work therein, we have studied a possible model of the syntax and semantics of the calculus, using the presheaf category $\text{Set}^E$, where $E$ is the category of equivalence relations and equivalence preserving morphisms.

Another case study is the NCP calculus defined in [CFGS08b]. States in the semantics of NCP are pairs consisting of a process, and the explicit network topology, in the form of a graph. We prove that the semantics is closed with respect to graph inclusion (nodes are fixed in NCP as they are names of global services, but arcs may appear and disappear, representing dynamic change of the topology). Noticeably, NCP features a network binding operation that allows to hide entire subnetworks. This seems very close to the case of pure names, but instead of having just a name, in this case we have network (sub)topologies attached to each state. Hence a natural presheaf model might be that of sets indexed by finite graphs. The translation of such a presheaf to (generalised) named sets is of interest in the light of generalising formal verification methods from nominal models to arbitrary presheaf categories.
Bibliography


History Dependent Automata for Service Compatibility, Lecture Notes in


[dB72] N. de Bruijn, Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem, Indag. Mat. 34 (1972), no. 5, 381–392. 13, 16, 38, 58


[Mil03] ——, *What’s in a name? (in honour of Roger Needham).* 11


