UNIVERSITÀ DEGLI STUDI DI PISA

DIPARTIMENTO DI INFORMATICA
DOTTORATO DI RICERCA IN INFORMATICA
SETTORE SCIENTIFICO DISCIPLINARE: INF/01

PH.D. THESIS

A Network-Aware Process Calculus for Global Computing and its Categorical Framework

Matteo Sammartino

SUPERVISOR

Ugo Montanari

REFEREE

Bartek Klin

REFEREE

Björn Victor

Abstract

An essential aspect of distributed systems is *resource management*, concerning how resources can be accessed and allocated. This aspect should also be taken into account when modeling and verifying such systems. A class of formalisms with the desired features are *nominal calculi*: they represent resources as atomic objects called *names* and have linguistic constructs to express creation of new resources. The paradigmatic nominal calculus is the π-calculus, which is well-studied and comes with models and logics.

The first objective of this thesis is devising a *natural* and *seamless* extension of the π-calculus where resources are network nodes and links. The motivation is provided by a recent, successful networking paradigm called Software Defined Networks, which allows the network structure to be manipulated at runtime via software. We devise a new calculus called *Network Conscious π-calculus* (NCPi), where resources, namely nodes and links, are represented as names, following the π-calculus guidelines. This allows NCPi to reuse the π-calculus name-handling machinery. The semantics allows observing end-to-end routing behavior, in the form of routing paths through the network. As in the π-calculus, bisimilarity is not closed under input prefix. Interestingly, closure under parallel composition does not hold either. Taking the greatest bisimulation closed under all renamings solves the issue only for the input prefix. We conjecture that such closure yields a full congruence for the subcalculus with only guarded sums.

We introduce an extension of NCPi (κNCPi) with some features that makes it closer to real-life routing. Most importantly, we add *concurrency*, i.e. multiple paths can be observed at the same time. Unlike the sequential version, bisimilarity is a congruence from the very beginning, due to the richer observations, so κNCPi can be considered the "right" version of NCPi when compositionality is needed. This extended calculus is used to model the peer-to-peer architecture `Pastry` [59].

The second objective is constructing a convenient operational model for NCPi. We consider *coalgebras*, that are categorical representation of system. Coalgebras have been studied in full generality, regardless of the specific structure of systems, and algorithms and logics have been developed for them. This allows for the application of general results and techniques to a variety of systems.

The main difficulty in the coalgebraic treatment of nominal calculi is the presence of name

binding: it introduces α-conversion and makes SOS rules and bisimulations non-standard. The consequence is that coalgebras on sets are not able to capture these notions. The idea of the seminal paper [28] is resorting to coalgebras on *presheaves*, i.e. functors $\mathbf{C} \to \mathbf{Set}$. Intuitively, presheaves allow associating to collections of names, seen as objects of $\mathbf{C}$, the set of processes using those names. Fresh names generation strategies can be formalized as endofunctors on $\mathbf{C}$, which are lifted to presheaves in a standard way and used to model name binding. Within this framework, a coalgebra for the π-calculus transition system is constructed: the benefit is that ordinary coalgebraic bisimulations for such coalgebra are π-calculus bisimulations. Moreover, [28] shows a technique to obtain a new coalgebra whose bisimilarity is closed under all renamings. This relation is a congruence for the π-calculus.

Presheaves come with a rich theory that can help deriving new results, but coalgebras on presheaves are impractical to implement: the state space can be infinite, for instance when a process recursively creates names. However, if we restrict to a class of presheaves (according to [17]), coalgebras admit a concrete implementation in terms of HD-automata [50], that are finite-state automata suitable for verification.

In this thesis we adapt and extend [28] to cope with network resources. First we provide a coalgebraic semantics for NCPi whose bisimulations are bisimulations in the NCPi sense. Then we compute coalgebras and equivalences that are closed under all renamings. The greatest such equivalence is a congruence w.r.t. the input prefix and we conjecture that, for the NCPi with only guarded sums, it is a congruence also w.r.t. parallel composition. We show that this construction applies a form of *saturation*, in the sense of [10]. Then we prove the existence of a HD-automaton for NCPi. The treatment of network resources is non-trivial and paves the way to modeling other calculi with complex resources.

Contents

Chapter 1

Introduction

1.1 Nominal calculi

An essential aspect of distributed systems is *resource management*, concerning how resources can be accessed and allocated. An example of popular distributed systems where this aspect is preponderant is *Cloud Computing*, whose main function is the remote, on-demand provisioning of various kinds of resources (storage, computational power, networking infrastructure,...) to users, who then do not need to maintain their own computing hardware and software.

Until the nineties, the available formalisms for modeling distributed systems, such as Hoare's CSP [37] or Milner's CCS [47], lacked constructs for expressing resource allocation: they could only model fixed communication infrastructure. An important breakthrough was made in 1992, when Milner et al. introduced the π-calculus [48]. This is based on the notion of *names*, that are entities characterized only by their identity, used to model *communication channels* along which processes can exchange messages. A key feature is *mobility*: messages are names themselves, so channels can move among processes, with the effect of reconfiguring the communication infrastructure. The additional expressive power with respect to previous calculi comes from the possibility of communicating local names.

Consider, for instance, the CCS process $(p \mid q)\backslash b$, denoting two parallel processes p and q that share a private port b. The π-calculus counterpart is $(b)(p \mid q)$: the *restriction operator* (b) binds b in both p and q and makes it unguessable by other processes in the environment. However b, like any other name, can be communicated. To see this, suppose that p has the form $\overline{a}b.p'$, meaning that it can send b along channel a, and suppose there is another process in the environment made of two components, one of which is waiting for some datum at a:

$$\underbrace{(b)(\overline{a}b.p' \mid q)}_{p_1} \;\mid\; \underbrace{(a(x).r \mid s)}_{p_2} \;.$$

Then, any input performed by $a(x).r$ is seen by p_1 as coming from the whole p_2. However, if

the received value is b, only r is entitled to use this channel after its reception, while s should not able to guess it. This is formally stated by requiring that b does not occur free in s. If this condition is satisfied, then b can be communicated, resulting in

$$(b)(\overline{a}b.p' \mid q \mid r[^{b}/_{x}] \mid s)$$

Notice that the communication of b caused its scope to be extended to p_2 (although s will not be able to use such channel). This mechanism is called *scope extrusion*.

The π-calculus is the forefather of a whole class of formalisms known as *nominal calculi*. Their common features are: representation of resources as names and the presence of linguistic constructs to express creation of new resources. Other examples of nominal calculi are: the *spi-calculus* [3], where encrypted values can be modeled, and passed in form of variables/names, and key generation as restriction; the *ambient calculus* [16], where names denote computational ambients that delimit the communication environment of processes, and restriction expresses the creation of a new ambient; the *applied π-calculus* [2], an extension of the π-calculus with the possibility of passing terms, generated from any given signature with equations; and others [58, 7, 12].

1.2 Coalgebraic models of nominal calculi

Category theory provides a convenient way of modeling the behavior of dynamic systems as *coalgebras* [60, 4], that are pairs (X, h) of a collection (not necessarily a set) of states X and a map h from each state to its possible evolution(s). The theory of coalgebras is rich and well-developed, and many kinds of systems have been characterized in this setting. For instance, finite branching LTSs on a set of labels L can be modeled as coalgebras of the form

$$X \longrightarrow \mathcal{P}_f(L \times X) \ , \tag{1.1}$$

mapping each state $x \in X$ to the finite set of pairs $(l, x') \in L \times X$ such that $x \xrightarrow{l} x'$.

Various notions have been captured at this abstract level. We have coalgebraic behavioral equivalences (see e.g. [64]) which, in the case of coalgebras of the form (1.1), coincide with standard ones for LTSs. We also have a characterization of *abstract semantics* and *minimal models*: the final object in a category of coalgebras, when it exists, describes the universe of all possible abstract behaviors, and the image of a coalgebra through the final morphism is its minimal version. The final coalgebra for (1.1) contains synchronization trees, and each state of a coalgebra is mapped to the tree representing its computation. Moreover, we have coalgebraic modal logics [55, 40], which include Hennessy-Milner logic [34] for coalgebras (1.1).

Coalgebras are also of practical interest: minimization procedures such as *partition refinement* [39], which are essential for finite-state verification, have been formulated in coalgebraic terms (see e.g. [5]). This further motivates the coalgebraic framework: algorithms implemented at this level of abstraction can be easily instantiated to many classes of systems.

Unfortunately, the operational semantics of nominal calculi does not easily fit into a coalgebraic description. This is mainly due to the presence of name binding operators in the syntax, which come with notions such as free and bound names, α-conversion, capture avoiding substitutions. In fact, SOS rules and bisimulations are subject to special side conditions, intended to enforce freshness of names. To see this, consider the following π-calculus SOS rule for parallel composition

$$\frac{p \xrightarrow{\mu} p'}{p \mid q \xrightarrow{\mu} p' \mid q} \; \mathrm{bn}(\mu) \cap \mathrm{fn}(q) = \varnothing$$

where the side condition is essential to ensure that the free names of q are not mistaken for bound names of the whole process, and thus possibly instantiated as a consequence of an interaction with the environment. As for bisimulations, they are not the ordinary ones for LTSs. For instance, every π-calculus bisimulation has the following freshness requirement:

> R is a bisimulation if, for each $(p, q) \in R$, whenever $p \xrightarrow{\mu} p'$, **with** $\mathrm{bn}(\mu)$ **fresh w.r.t.** q...

The reason is that $\mathrm{bn}(\mu)$ can be regarded as a variable (due to α-conversion), so its identity is irrelevant and should be chosen in a way that does not affect the ability of q to simulate the transition of p.

On the practical side, model-checking is quite problematic: the transition system can be infinite-branching, because all the possible instances of bound names in transition labels must be considered; moreover, a loop generating new names can make the number of states infinite.

All these issues arise if one attempts a set-based approach to the semantics of nominal calculi, for instance employing coalgebras of the form (1.1). A more satisfactory approach has been devised by Fiore and Turi in [28] for the paradigmatic case of the π-calculus. The idea is to make coalgebras "resource-aware" by considering *processes in contexts* $C \vdash p$, where C is an over-approximation of the amount of resources owned by p (its free names, in the case of the π-calculus).

We sketch the categorical implementation of this idea. Given a category $\mathbf{C}$ of collections of resources, processes in contexts are formalized as a *presheaf*, that is a functor $P \colon \mathbf{C} \to \mathbf{Set}$ giving the set of processes indexed by each $C \in |\mathbf{C}|$. The category $\mathbf{C}$ is equipped with *allocation operators* $\delta \colon \mathbf{C} \to \mathbf{C}$, adding a unique fresh resource to $C \in |\mathbf{C}|$. This new resource canonically represents all the ones fresh with respect to C.

The operational semantics is modeled as a coalgebra on the presheaf of processes. This coalgebra can be understood as a transition system on indexed states, of the form

$$C \vdash p \xrightarrow{\mu} C' \vdash p'$$

Allocation transitions of the ordinary transition system, when they differ only for the identity of the communicated fresh names, can all be represented as a *unique* indexed

transition where $C' = \delta C$, for a suitable allocation operator δ, and $\mathrm{bn}(\mu)$ is exactly the new resource in δC. This representation makes the model finite-branching. As for bisimulations, processes are compared for equivalence only if they have the same amount of resources C: freshness requirements become useless, because the only possible fresh name is the new one in δC.

Unfortunately, the state space explosion issue still exists: there is no way of deallocating resources along transitions, so a recursive process may give rise to an infinite chain of transitions

$$C \vdash p \xrightarrow{\mu} C' \vdash p' \xrightarrow{\mu'} C'' \vdash p'' \xrightarrow{\mu'''} \dots$$

with $C \subset C' \subset C'' \subset \dots$, even if some of the reached processes are not actually using the new resources. However, if the presheaf of states is "well-behaved", according to [17], it is always possible to compute the minimal amount of resources a process uses. This is the key condition for the equivalence between presheaf-based coalgebras and *History Dependent (HD) automata* [50], that are automata with allocation and deallocation along transitions. HD automata modeling *finite-control processes*, i.e. processes without parallel composition inside recursions, admit minimal representatives, where all bisimilar states are identified. These can can be computed as shown and implemented in [25].

It is worth pointing out that [28] is not the first model of resource handling based on functor categories: the denotational semantics of variable allocation in block-structured languages (ALGOL) has been given in terms of functors from a category of store shapes and store expansions [56].

1.3 Subject of this thesis

The trend in networking is going towards more "open" architectures, where the infrastructure can be manipulated in software. This trend started in the nineties, when OpenSig [14] and Active Networks [65] were presented, but neither gained wide acceptance due to security and performance problems. More recently, OpenFlow [45, 1] or, more broadly, Software Defined Networking has become the leading approach, supported by Google, Facebook, Microsoft and others.

Software defined networks (SDNs) allow network administrators to control traffic via software installed on a centralized *controller*. This machine is connected to all switches in the network, and instructs them to install or uninstall forwarding rules and report traffic statistics. The single most important function of SDNs is controlling *flows*, that are the routing paths of data characterized by specific properties, e.g. certain values of QoS parameters.

Traditional process calculi, such as π-calculus [48], CCS [47] and others, seem inadequate to describe these kinds of networks. In fact, they abstract away from network details, as two processes are allowed to communicate only through shared channels. Complex

infrastructural elements, such as network links, could be described in terms of processes, and routing protocols in terms of consecutive step-by-step forwardings. However, end-to-end routing behavior could not be observed in a single transition, e.g. as the the path for a SDN flow. This information can be useful for the analysis of routing algorithms, e.g. to determine whether they are always able to construct a valid/optimal path for given source and destination.

To give better visibility to the network architecture, in recent years network-aware extensions of known calculi have been devised [29, 21]. However, they do not allow observing multi-hop routing paths, which then are not taken into account by bisimulations, and their underlying calculi have not been studied from a coalgebraic perspective.

1.3.1 A network-conscious π-calculus

In this thesis we introduce the *Network Conscious π-calculus* (NCPi), a seamless extension of the π-calculus with a natural notion of network: nodes and links are regarded as computational resources that can be created, passed and used to transmit, so they are represented as names, following the π-calculus methodology. The main features of NCPi are the following:

- There are two types of names: *sites*, which are the nodes of the network, and *links*, named connectors between pairs of sites. Sites are not locations where processes run, like in [29], but they can be regarded as *network interfaces*. In fact, we allow each process to use more than one site. Both are atomic names, but links are equipped with source and target operations that give their endpoints; for convenience we write l_{ab} for a link l from a to b. Free sites and links of a process form the network it can use to communicate. The structure of this network should be respected by renamings, so we only allow for *graph homomorphisms*.

- The syntax can express the *creation* of a link through the restriction operator, and the *activation* of a transportation service over a link through a dedicated prefix. Separating these operations agrees with the π-calculus, where creating and using a channel as subject are two distinct operations. This is different from [29, 21], where pieces of network, once created, are always available. In fact, in [29] the connectivity of a process P is described as a context $\Delta \rhd P$, which can be augmented through scope extrusion. In [21] there are special processes $\{l_1 \leftrightarrow l_2\}$ representing bidirectional connections between l_1 and l_2, which can be used by the processes in parallel with them. There is a dedicated primitive to spawn link processes.

- Observations of the labelled semantics represent transmissions in the form of routing paths, i.e. sequences of links used for communications. This, as mentioned, is motivated by the possibility of modeling and analyzing routing algorithms.

We choose to have named links, whereas in [29] and [21] links are anonymous pairs of locations. There are two main reasons for our choice. First of all, this allows distinguishing two links between the same pair of nodes, which could represent connections with different features (cost, bandwidth...). These pieces of information could be associated to links via suitable operations, as it happens for source and target, and used to model routing that depends on quantities, e.g. QoS-based routing. Second, this enables reusing most of the notions of the π-calculus (renaming, α-conversion, extrusion...), suitably extended. In any case, NCPi allows one to recover anonymous connectors through the restriction operator.

One of the non-trivial aspects of NCPi is the presence of parametric names: links, in fact, are parametrized by other names, i.e. their endpoints. While this is a natural choice, it requires some care. Consider in fact a process p with a free occurrence of a link l_{ab}. If we render a unobservable, as in $(a)p$, such link would appear "dangling" to an external observer. A first solution is establishing that the link is private as well. A more elegant solution is identifying a class of well-behaved processes, where only private links can have private endpoints, i.e. $(a)(l_{ab})p$ is correct. Notice, however, that $(l_{ab})p$ is also correct: it means that the link is private, but it still can be used to connect public sites.

We remark that also the ψ-calculus [7] and the π-calculus with polyadic synchronization [15] can represent connectors, respectively as nominal terms and tuples of names. The former representation is closer to ours, while the latter is too "flat", in the sense that it does not capture the graph-structure of network and the fact that renamings should respect it. Moreover, the semantics of connectors is different. In [15] and [7] they are used as π-calculus channels, i.e. synchronization involves two processes, and happens on shared, or equivalent, connectors, which are then rendered unobservable. In our calculus, instead: communications involve synchronizations between many processes, namely those that are willing to communicate and those that provide links, which cooperate to construct a routing path; synchronizations can happen on the two endpoints of a link, separately; (global) links are always observable when used in a communication.

The last aspect we discuss is congruence. As in the π-calculus, bisimilarity is not closed under input prefix. The interesting fact is that it not closed under parallel composition, either. The intuition is that adding router processes could allows some processes to send data through longer paths. We conjecture that, for NCPi processes with guarded sums, the greatest bisimulation closed under all renamings is a congruence with respect to input prefix, as in the π-calculus, and to parallel composition. This issue completely disappears for the concurrent version of NCPi that will be described later.

1.3.2 NCPi categorical operational semantics

We construct two presheaf-based coalgebraic operational models for NCPi, one for observational equivalence and one, closed under all renamings, whose equivalence is closed under input prefix and is conjectured to be a full congruence for NCPi with guarded sums.

Our approach follows and generalizes the one of [28]. The main novelty is the treatment of complex resources, namely communication networks, where some names (links) are parametrized by other names (sites).

Here is an overview of our approach. We represent communication networks as a category of finite, directed multigraphs, equipped with two allocation operators that create fresh vertices and edges. Then, we construct a category of coalgebras on presheaves indexed by such graphs, where allocation happens according to the allocation operators. In this category, we model the NCPi operational semantics and characterize its observational equivalences. Then, employing categorical constructs called *right Kan extensions*, we transfer the semantics to another category of coalgebras, gaining closure under all renamings. These, roughly, are the steps described in [28].

We go a little further: we give an explicit characterization of coalgebras in the last category as *saturated* indexed transition systems [10], and we show that right Kan extension performs saturation. Finally, we show that the categorical operational semantics of our calculus can be implemented as a HD-automaton.

The purpose of this construction is: on the one hand, validating the technique of [28] on a calculus with much richer resources than atomic channels; on the other hand, integrating different models that take this additional complexity into account, such as presheaf-based coalgebras, saturated transition systems and HD-automata. This paves the way for the treatment of other complex calculi.

1.3.3 A concurrent semantics for NCPi

Interleaving semantics can be considered inadequate for distributed systems with partially asynchronous behavior, because reducing parallelism to sequential nondeterminism can be regarded as imposing a centralized resource manager that grants access to resources in some order. This criticism is particularly relevant for NCPi, because it aims at modeling systems that can easily have a high degree of parallelism, therefore the presence of a centralized resource manager is unrealistic.

Our answer is an extended version of NCPi, called *concurrent NCPi* (κNCPi), where observations include additional routing information and are *multisets* of routing paths. The concurrent nature of the semantics allows distinguishing concurrent from interleaving behavior, in fact we have (using a π-calculus notation)

$$\overline{a}c.\mathbf{0} \mid b(x).\mathbf{0} \xrightarrow{\overline{a}c \mid b(x)} \mathbf{0} \qquad \overline{a}c.b(x).\mathbf{0} + b(x).\overline{a}c.\mathbf{0} \not\xrightarrow{\overline{a}c \mid b(x)}$$

whereas they are bisimilar in the π-calculus. This causes bisimilarity not to be closed under input prefix. In fact, if we prefix both processes with $d(a)$ we get:

$$d(a).(\overline{a}c.\mathbf{0} \mid b(x).\mathbf{0}) \xrightarrow{db} \overline{b}c.\mathbf{0} \mid b(x).\mathbf{0} \xrightarrow{\tau} \mathbf{0}$$

$$d(a).(\overline{a}c.b(x).\mathbf{0} + b(x).\overline{a}c.\mathbf{0}) \xrightarrow{db} \overline{b}c.b(x).\mathbf{0} + b(x).\overline{b}c.\mathbf{0} \not\xrightarrow{\tau}$$

A suitable syntactic restriction of κNCPi will allow us to equip the π-calculus with a concurrent semantics whose bisimilarity is a congruence.

1.3.4 `Pastry` model

A peer-to-peer system provides the networking substrate for the execution of distributed applications. It is made of *peers* that interact over an application-level *overlay network*, built on top of the physical one. An overlay network is highly dynamic, as peers can join and leave it at any time, and this causes continuous reconfigurations of its topology.

The dynamic nature of peer-to-peer overlay networks makes them an interesting case study for our calculus. Our reference architecture will be `Pastry` [59]. In `Pastry`, peers have unique identifiers, logically ordered in a ring. The main operation is routing by key: given a message and a target key, the message is delivered to the peer whose identifier is numerically closest to the key. `Pastry` is typically used for implementing *Distributed Hash Tables* (DHT), that are hash tables whose entries are distributed among peers: routing by key in this context amounts to hash table lookup.

Our `Pastry` model is as follows. We begin by formalizing the features of `Pastry` routing that ensure its convergence. These are informally stated in [59], but we need a rigorous formulation so that we can prove the correctness of our model. Then we give a κNCPi implementation of a `Pastry` peer. The basic idea is capturing the overlay as a collection of links between peers. Our implementation of a `Pastry` peer has two functions: handling of node joins and provision of routing services to applications. Node joins trigger a complex procedure, ending up with the creation of new links from/to the joining peer. We show that the resulting overlay still guarantees routing convergence. Finally, we model a simple DHT, where lookups are represented as routing paths from the peer that invoked the lookup to the one responsible for the target key. These are derived by composing atomic forwarding services provided by peers. We prove that we have routing convergence also in this scenario, i.e. lookups always reach the correct peer.

1.4 Contributions

We summarize the main contributions of this thesis. Some are developments of results that have already been published in three works by the author:

1. [52], published in the proceedings of *Mathematical Foundations of Programming Semantics* XXVIII;
2. [53], to appear in *Theoretical Computer Science*;
3. a technical report [51].

In the following we will explicitly cite the main source for each contribution. The `Pastry` model is unpublished.

A seamless extension [53]. We introduce a network-aware extension of the π-calculus which is *seamless*, in the sense that we reuse concepts and mechanisms of the π-calculus to represent and operate on network resources. In particular, we have that links, like π-calculus channels, can be explicitly activated, whereas networks are always active in [29, 21]. Our networks are more *programmable*, in the sense that we can program creation, activation and usage of network resources.

Non-compositionality of interleaving routing. The fact that bisimulation for NCPi is not closed under parallel composition is an interesting result. It essentially says that routing is intrinsically non-compositional, if one considers interleaving computations. In fact, this issue disappears for κNCPi, where parallelism can be explicitly observed.

Complex resources [53]. We study resources that are considerably more complex than π-calculus channels. The main difficulty is that we have parametric names. At the linguistic level, this may produce meaningless process expressions, so we have to identify a class of *well-formed processes*. At the level of models, we represent such names as graphs, which are important and well-studied in computer science; the novelty (to the best of our knowledge) is using graphs as index category for presheaves with the purpose of modeling allocation of network resources. We manage to give a formulation of allocation operations in a way that confers good properties to coalgebras using them: the existence of a final coalgebra and the fact that minimal coalgebras are indeed those quotiented by bisimilarity. Since graphs are widely used, such an abstract and well-behaved notion of allocation for graphs could be useful in other contexts.

Saturation as Right Kan extension [53]. We develop the characterization of saturation as right Kan extension discussed in [8]. In particular, we give an explicit characterization of extended coalgebras in terms of *saturated* indexed transition systems. This has been done also in [8], but in a concise way. The novelty is that we consider more complex categories, prove some additional results (e.g. existence of the final coalgebra) and we consider coalgebras over *sheaves*, unlike both [28] and [8]. The scope of these results is potentially broader than the specific incarnation treated in this thesis, as saturation is a well-known and general technique to get closures under many kinds of contexts (see e.g. [54]).

History Dependent Automata [53]. We show that the NCPi coalgebraic semantics can be implemented as a HD-automaton, which could allow checking properties of finite-control NCPi processs in an efficient way. This result follows from various properties of our index category for presheaves and of the specific presheaf of processes we employ, the most important being the possibility of computing *minimal support*, i.e. the minimal network a process uses. Thanks to this property, bisimilar processes with the same minimal network

can be collapsed to a canonical representative, with the consequence that the state space becomes finite. This is essential for verification.

Congruence via concurrency [52]. We prove that observing concurrency makes bisimilarity for our calculus a congruence. This is a desirable property for a process calculus, because it allows for the compositional analysis of systems and enables categorical models such as *bialgebras* [66] without resorting to saturation.

The authors of [21, 29] treat bisimilarity and achieve compositionality for their network-aware calculi. Their approach, considered standard in the literature, is the following: they start from a reduction semantics, guess a suitable notion of barb, define barbed congruence by closing w.r.t. all the contexts, and then characterize it as a bisimulation equivalence on a labelled version of the transition system. In general, this approach yields labelled transition systems with succinct observations, but requires a reduction semantics and may produce a non-standard notion of bisimilarity, where closure under contexts is enforced in the definition. For instance, the notion of bisimulation of [21] requires a process to be embedded in a bigger network, which in some sense provides the missing facilities, in order to be compared with a process that exhibits a certain kind of label. We show that we can gain the congruence property through a concurrent labelled semantics, while keeping the notion of bisimilarity as standard as possible. We argue that a barb-based approach for κNCPi would not be of particular interest: barbs would be inputs and outputs without links, and the missing network would be provided by contexts, as in [21].

We exploit this result to equip the π-calculus with a concurrent and compositional semantics. The π-calculus, in fact, can be easily characterized as a syntactic restriction of κNCPi, where sites play the role of channels and links are forbidden. This shows that bisimilarity not being a congruence for the ordinary π-calculus depends on the interleaving nature of the semantics, and not on the language itself. An analogous result is [42, 43], but the semantics presented there allows observing the channel where a synchronization is performed, whereas our concurrent semantics for the π-calculus is more faithful, in the sense that we adopt a synchronization mechanism that hides such a channel. A comparison with other concurrent semantics for the π-calculus can be found in §7.1.3.

A significant case study. Our model of `Pastry` gives an evidence of how expressive κNCPi can be. In fact, the synchronization mechanism allows deriving the complete routing path for a given key, in the form of a sequence of links from the sender to the key's owner. While an implementation in the π-calculus is feasible, where links are modeled as forwarding processes, this would not yield the same semantics: each transition would allow observing one single-hop communication.

1.5 Map and origin of the chapters

This thesis is organized into five chapters. We give a brief synopsis of each one.

Chapter 2: Background. In this chapter we survey some background needed for the rest of the thesis. First, we recapitulate the π-calculus syntax and semantics. Then we introduce some category-theoretic concepts such as presheaves and sheaves, locally presentable categories, accessible functors, Kan extensions, coalgebras and coalgebras over presheaves. Finally, we give an overview of the model [28], with the purpose of highlighting the important features of presheaf-based coalgebraic models of nominal calculi.

Chapter 3: Network-Conscious π-calculus. In this chapter we introduce NCPi. We begin with an illustrative example, aimed at showing the capability of our language to model networks with reconfigurable structure. Then we describe the syntax and semantics of the calculus, and we list some relevant properties that make the transition system and its bisimulations well suited for a coalgebraic representation. Next we discuss the congruence property: this does not hold for the full calculus, even considering bisimulations that are closed under all renamings, but we hint at a syntactic restriction that could enable congruence. Finally, we demonstrate the convenience of our calculus by making a comparison with the π-calculus. This chapter is the revised and extended version of [53, §3].

Chapter 4: Coalgebraic Semantics of NCPi. We begin with an overview of the general approach, inspired by [28]. Then we systematically apply it to model the operational semantics of NCPi. Here the main results are: Theorem 4.3.10, which provides a coalgebraic characterization of observational equivalence; Theorem 4.5.7, which does the same for the greatest equivalence closed under all renamings, thus under input prefix; §4.4, where we discuss the existence of HD-automata for NCPi. This chapter comes from [53, §4].

Chapter 5: Concurrent NCPi. In this chapter we extend the NCPi syntax and semantics with new constructs and observations, with the purpose of describing a more realistic network behavior. The main result is Theorem 5.2.7, stating that concurrent bisimilarity is a congruence. Finally, we give a simple model of the *Border Gateway Protocol*. The main source for this chapter is [51], summarized in [53, §5].

Chapter 6: Case study: `Pastry` . In this chapter we use κNCPi to model two aspects of the peer-to-peer protocol `Pastry`: reconfiguration due to node joins and routing. We formalize convergence of routing in `Pastry` as Property 6.1.2, and we prove that node joins preserve this property (Theorem 6.3.1). Then we give a simple model of a Distributed Hash Table, equipped with a semantics that captures routing of DHT key lookups. Theorem 6.4.2 states that routing converges also in this scenario.

Chapter 2

Background

2.1 The π-calculus

In this section we recall the syntax and semantics of the π-calculus. Our main references are [57, 48, 62].

2.1.1 Syntax

We assume a countable set of *names* $\mathcal{N}$, ranged over by $a, b, x, y, \ldots$ These denote communication channels, and are the building blocks of processes.

Definition 2.1.1 (Processes)**.** π-calculus processes are generated by the following grammar

$$\begin{array}{lll} p & ::= & \mathbf{0} \mid \pi.p \mid p + p \mid p \,|\, p \mid (x)p \mid \mathtt{A}(y_1, \ldots, y_n) \\ \pi & ::= & \overline{a}x \mid a(x) \mid \tau \end{array}$$

$$\mathtt{A}(x_1, \ldots, x_n) \stackrel{\text{def}}{=} p \qquad i \neq j \implies x_i \neq x_j$$

Process terms have the following meaning: $\mathbf{0}$ is the process that cannot perform any action, $p + q$ is the process behaving non-deterministically as p or q, $p \,|\, q$ denotes the concurrent execution of p and q, $(x)p$ is the process where x is private to p, and $\mathtt{A}(y_1, \ldots, y_n)$ can be thought of as the invocation of a process previously defined using $\stackrel{\text{def}}{=}$, where the formal parameters $x_1, \ldots, x_n$ have been replaced by the formal ones $y_1, \ldots, y_n$. A prefix π can be:

- the *output prefix* $\overline{a}x.p$, meaning that the process can send x on channel a and continue as p;
- the *input prefix* $a(x).p$, meaning that the process can input some name, to be bound to x, on channel a and continue as p;

α-conversion:

$$(x)p \equiv (y)p[y/x] \qquad a(x).p \equiv a(y)p[y/x] \qquad\qquad y \# p$$

Commutative monoidality of $\mid$ and $+$:

$$\begin{aligned} p \mid \mathbf{0} &\equiv p & p + \mathbf{0} &\equiv p \\ p_1 \mid p_2 &\equiv p_2 \mid p_1 & p_1 + p_2 &\equiv p_2 + p_1 \\ p_1 \mid (p_2 \mid p_3) &\equiv (p_1 \mid p_2) \mid p_3 & p_1 + (p_2 + p_3) &\equiv (p_1 + p_2) + p_3 \end{aligned}$$

Scope extension

$$p_1 \mid (x)p_2 \equiv (x)(p_1 \mid p_2) \qquad\qquad x \# p_1$$

Unfolding law

$$\mathtt{A}(y_1, \ldots, y_n) \equiv p[y_1/x_1, \ldots, y_n/x_n] \qquad \mathtt{A}(x_1, \ldots, x_n) \stackrel{\text{def}}{=} p$$

Figure 2.1: Structural congruence for π-calculus processes.

- the *silent prefix* $\tau.p$, meaning that the process can perform an internal, unobservable action denoted by τ and continue as p.

We denote by $\mathrm{n}(p)$ the set of names occurring in p, and by $\mathrm{fn}(p)$ the *free names* of p, defined as follows

$$\begin{aligned} \mathrm{fn}(\mathbf{0}) &:= \varnothing & \mathrm{fn}(\overline{a}x.p') &:= \{a, x\} \cup \mathrm{fn}(p') \\ \mathrm{fn}(a(x).p') &:= \{a\} \cup (\mathrm{fn}(p') \setminus \{x\}) & \mathrm{fn}((x)p) &:= \mathrm{fn}(p) \setminus \{x\} \\ \mathrm{fn}(p_1 + p_2) &:= \mathrm{fn}(p_1 \mid p_2) := \mathrm{fn}(p_1) \cup \mathrm{fn}(p_2) & \mathrm{fn}(\tau.p') &:= \mathrm{fn}(p') \end{aligned}$$

$$\mathrm{fn}(\mathtt{A}(x_1, \ldots, x_n)) := \{x_1, \ldots, x_n\}$$

and we must have $\mathrm{fn}(p) \subseteq \{x_1, \ldots, x_n\}$ whenever $\mathtt{A}(x_1, \ldots, x_n) \stackrel{\text{def}}{=} p$. The bound names $\mathrm{bn}(p)$ are given by

$$\mathrm{bn}(a(x).p') := \mathrm{bn}((x)p') := \{x\} \cup \mathrm{bn}(p')$$

and as the union of the bound names of the operator's arguments in all the other cases. In the following we write $x \# p$ to mean that x is fresh with respect to p, i.e. $x \notin \mathrm{fn}(p)$; this notation is extended to sets of names N, namely $N \# p$, with the expected meaning.

Now we recall the notion of *renaming*.

Definition 2.1.2 (Renaming)**.** A renaming is a function $\sigma\colon \mathcal{N} \to \mathcal{N}$. We denote by $[y_1/x_1, \ldots, y_n/x_n]$ the renaming that replaces x_1 with y_1, …, x_n with y_n. Given a process p, $p\sigma$ denotes the result of applying σ to the free names of p, in a way that avoids captures.

A renaming is said to be capture avoiding if it never replaces free occurrence of names with bound ones. One way to guarantee this is via α-conversion: if σ performs $[y/x]$ and p has $(y)p'$ or $a(y).p'$ as subprocess, with $x \in \mathrm{fn}(p')$, then σ is applied to an α-converted version of p, where y has been replaced with a fresh name, so that y is free in $p\sigma$. The formal definition of α-conversion is given in Figure 2.1 and, as usual, is "mutually recursive" with that of renaming.

The π-calculus processes come equipped with a structural congruence relation $\equiv$, defined in Figure 2.1, telling which processes should be considered equivalent. Besides α-conversion, there are other axioms stating that: the order of processes and how they are associated in a parallel composition or sum does not count; we can always bring a restriction outside a parallel composition, provided that in doing so the restriction does not capture free occurrences of its argument; a process definition invocation corresponds to replacing formal parameters with actual ones in its body.

2.1.2 Operational semantics

The standard π-calculus semantics is defined as a labelled transition system, derived through structural operational semantics rules. We have five kinds of labels:

- the *internal action* τ, meaning that an unobservable action is performed;
- the *free input action* ab, meaning that the value b is received along channel a;
- the *bound input action* $a(x)$, meaning that some value is received along channel a, but its value will be bound to x as soon as it becomes known;
- the *free output action* $\overline{a}b$, meaning that the name b is emitted along channel a;
- the *bound output action* $\overline{a}(x)$, meaning that the local name x is published along channel a; this is called *extrusion*.

Given an action μ, its *subject* $\mathrm{subj}(\mu)$, *object* $\mathrm{obj}(\mu)$, *free* names $\mathrm{fn}(\mu)$ and *bound* names $\mathrm{bn}(\mu)$ are defined in Table 2.1.

The *early* and *late* π-calculus transition systems, denoted respectively by $\longrightarrow_e$ and $\longrightarrow_l$ (when no subscript is specified, the transition can be of either kind), are the smallest ones generated by the rules in Figure 2.2. "Early" and "late" refer to the moment the input placeholder is instantiated with an actual value: (IN_e) instantiates it as soon as the input prefix is consumed, producing a free input action, while (IN_l) leaves the placeholder symbolic, yielding a bound input action. This distinction gives rise to two different rules for free names communication: (COM_e) requires that the input object in the premises is chosen to

$$\text{(OUT)}\quad \frac{}{\overline{a}b.p \xrightarrow{\overline{a}b} p} \qquad \text{(INT)}\quad \frac{}{\tau.p \xrightarrow{\tau} p}$$

$$\text{(RES)}\quad \frac{p \xrightarrow{\mu} p'}{(x)p \xrightarrow{\mu} (x)p'}\ x \notin \mathrm{n}(\mu) \qquad \text{(OPEN)}\quad \frac{p \xrightarrow{\overline{a}x} p'}{(x)p \xrightarrow{\overline{a}(x)} p'}\ a \neq x$$

$$\text{(SUM)}\quad \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \qquad \text{(PAR)}\quad \frac{p \xrightarrow{\mu} p'}{p \mid q \xrightarrow{\mu} p' \mid q}\ \mathrm{bn}(\mu) \# q$$

$$\text{(STRUCT)}\quad \frac{p \equiv p' \quad p' \xrightarrow{\mu} q' \quad q' \equiv q}{p \xrightarrow{\mu} q}$$

Early semantics

$$\text{(IN}_e\text{)}\quad \frac{}{a(x).p \xrightarrow{ab}_e p[b/x]} \qquad \text{(COM}_e\text{)}\quad \frac{p \xrightarrow{\overline{a}b}_e p' \quad q \xrightarrow{ab}_e q'}{p \mid q \xrightarrow{\tau}_e p' \mid q'}$$

Late semantics

$$\text{(IN}_l\text{)}\quad \frac{}{a(x).p \xrightarrow{a(x)}_l p} \qquad \text{(COM}_l\text{)}\quad \frac{p \xrightarrow{\overline{a}b}_l p' \quad q \xrightarrow{a(x)}_l q'}{p \mid q \xrightarrow{\tau}_l p' \mid q'[b/x]}$$

Figure 2.2: Structural operational semantics rules for the π-calculus.

μ	$\mathrm{subj}(\mu)$	$\mathrm{obj}(\mu)$	$\mathrm{fn}(\mu)$	$\mathrm{bn}(\mu)$
$\overline{a}b$	a	b	$\{a, b\}$	$\varnothing$
$\overline{a}(x)$	a	x	$\{a\}$	$\{x\}$
ab	a	b	$\{a, b\}$	$\varnothing$
$a(x)$	a	x	$\{a\}$	$\{x\}$
τ	$-$	$-$	$\varnothing$	$\varnothing$

Table 2.1: Notation for π-calculus actions.

coincide with the output object; (COM$_l$) does not impose such requirement, but instantiates the input placeholder in the inferred continuation.

We briefly recall what the other rules mean. (OUT) infers an output. (RES) and (OPEN) turn an action μ of p into one of $(x)p$: (RES) is applied when x does not appear in μ, and binds x in the continuation while leaving the action unchanged; (OPEN) is applied when μ is a free output with object x, and binds x in the action but not in the continuation, meaning that x becomes global after the transition. In other terms, bound output actions express the allocation of a fresh name. (SUM) states that the sum of two processes behaves as either of them. (PAR) states that, if we put in parallel two processes, one of which stays idle, the actions of the overall process are those of the active one, provided that the bound names in these actions are fresh w.r.t. the other process. Finally, (STRUCT) allows one to use structural congruence during inference.

Scope extension axioms can be replaced by a rule (CLOSE), dual to (OPEN), which extends the scope of a bound name as effect of its communication:

$$\text{(CLOSE)} \quad \frac{p \xrightarrow{a(x)} p' \quad q \xrightarrow{\overline{a}(x)} q'}{p \mid q \xrightarrow{\tau} (x)(p' \mid q')}$$

This requires including a bound input action also in the early transition system, specifically intended for the reception of bound names.

Now we list some properties of the transition systems.

Proposition 2.1.3. *Transitions are* reflected *and* preserved *by any injective substitution σ, explicitly:*

- *if $p \xrightarrow{\mu} p'$ then $p\sigma \xrightarrow{\mu\sigma} p'\sigma$ (preservation);*
- *if $p\sigma \xrightarrow{\mu} p'$ then there are μ' and p'' such that $\mu = \mu'\sigma$, $p' \equiv p''\sigma$ and $p \xrightarrow{\mu'} p''$ (reflection).*

Remark 2.1.4. Preservation also holds for generic substitutions, while reflection does not. In fact, consider the process

$$p \stackrel{\text{def}}{=} \overline{a}b.\mathbf{0} \mid c(x).\mathbf{0}$$

and the renaming $\sigma = [a/c]$. Then we have

$$p\sigma = \overline{a}b.\mathbf{0} \mid a(x).\mathbf{0} \xrightarrow{\tau} \mathbf{0}$$

but $p \not\xrightarrow{\tau} \mathbf{0}$.

2.1.3 Behavioral equivalences

There are various notions of bisimulation for the π-calculus. Here we recall the fundamental ones.

Definition 2.1.5 (Early and late bisimulations)**.** A binary, symmetric and reflexive relation $\mathcal{R}$ is a (early/late) bisimulation if, for all $(p, q) \in \mathcal{R}$:

(i) *Early bisimulation*: $p \xrightarrow{\mu}_e p'$, with $\mathrm{bn}(\mu)$ fresh w.r.t. q, implies that there is q' such that $q \xrightarrow{\mu}_e q'$ and $(p', q') \in \mathcal{R}$.

(ii) *Early bisimulation with late semantics*: $p \xrightarrow{\mu}_l p'$, with $\mathrm{bn}(\mu)$ fresh w.r.t. q, implies the following:

  (a) if $\mu = a(x)$ then, for all $y \in \mathcal{N}$, there is q' such that $q \xrightarrow{a(x)}_l q'$ and $(p'[y/x], q'[y/x]) \in \mathcal{R}$;

  (b) otherwise there is q' such that $q \xrightarrow{\mu}_l q'$ and $(p', q') \in \mathcal{R}$.

(iii) *Late bisimulation*: $p \xrightarrow{\mu}_l p'$, with $\mathrm{bn}(\mu)$ fresh w.r.t. q, implies the following:

  (a) if $\mu = a(x)$ then there is q' such that $q \xrightarrow{a(x)}_l q'$ and $(p'[y/x], q'[y/x]) \in \mathcal{R}$, for all $y \in \mathcal{N}$;

  (b) otherwise there is q' such that $q \xrightarrow{\mu}_l q'$ and $(p', q') \in \mathcal{R}$.

The greatest such relations are called *bisimilarities*, and are denoted by $\sim_e$, $\sim_e^l$ and $\sim_l$, respectively.

A relation being "early" or "late" depends on whether the input value is chosen before or after the continuation for q. In the early semantics this always happens before, according to the early SOS rules. In the late semantics we can have both situations: the difference is the relative order of the universal quantifier, ranging over all instantiations, and of the existential quantifier, expressing the existence of the continuation. The two notions of early bisimulations yield the same relations.

Proposition 2.1.6. *Early and late bisimilarities are closed under injective renamings, i.e. for all injective renamings σ we have that $p \sim q$ implies $p\sigma \sim q\sigma$, where $\sim$ is any of the mentioned bisimilarities.*

Now we recall the concept of *congruence*.

Definition 2.1.7 (Congruence)**.** A relation R is a congruence if, for any π-calculus operator op, $(p,q) \in R$ implies $(op(p), op(q)) \in R$.

Theorem 2.1.8. *$\sim_e$,$\sim_l$ and $\sim_e^l$ are congruences w.r.t. all operators except the input prefix.*

To see this in the case of early bisimilarity (other cases are analogous), consider the following processes (we omit objects as they are not relevant)

$$p \stackrel{\text{def}}{=} \overline{a} \mid b \qquad q \stackrel{\text{def}}{=} \overline{a}.b + b.\overline{a} \ .$$

We have $p \sim_e q$ but $c(a).p \not\sim_e c(a).q$, because

$$c(a).(\overline{a} \mid b) \xrightarrow{cb} \overline{b} \mid b \xrightarrow{\tau} \qquad c(a).(\overline{a}.b + b.\overline{a}) \xrightarrow{cb} \overline{b}.b + b.\overline{b} \not\xrightarrow{\tau} \ .$$

However, bisimulations that also are congruences have the following characterization.

Definition 2.1.9 (Early and late congruence)**.** Early (resp. late) congruences are those early (resp. late) bisimulations closed under all renamings.

Closure under all renamings implies closure under the input prefix, so these relations indeed are congruences.

2.2 Categorical notions

In this section we recall some notions of category theory that will be needed for the rest of the thesis: (pre)sheaves, locally presentable categories and accessible functors, Kan extensions, generic coalgebras and coalgebras over presheaves.

2.2.1 Presheaves and sheaves

Definition 2.2.1 (Functor category)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two categories. The *functor category* $\mathbf{B}^{\mathbf{A}}$ has functors $\mathbf{A} \to \mathbf{B}$ as objects and natural transformations between them as morphisms.

Our development will be based on a class of functors, called *presheaves*. These are functors from any category $\mathbf{C}$ to $\mathbf{Set}$. A presheaf P can be intuitively seen as a family of sets indexed over the objects of $\mathbf{C}$ plus, for each $f : c \to c'$ in $\mathbf{C}$, an action of f on Pc, which we write

$$p[f]_P := Pf(c) \qquad (p \in Pc) \ .$$

We will omit the subscript when clear from the context. The set $\int P$ of *elements* of a presheaf P is

$$\int P := \sum_{c \in |\mathbf{C}|} Pc$$

and we denote by $c \vdash p$ a pair belonging to $\int P$.

Recall that, given two categories $\mathbf{J}$ and $\mathbf{C}$, a *diagram of type* $\mathbf{J}$ *in* $\mathbf{C}$ is a functor $D\colon \mathbf{J} \to \mathbf{C}$, and $\mathbf{C}$ is said to have limits of type $\mathbf{J}$ whenever all diagrams of type $\mathbf{J}$ in $\mathbf{C}$ have a limit. For $\mathbf{C}$ with this property, there is a functor that computes such limits:

$$Lim\colon \mathbf{C}^{\mathbf{J}} \to \mathbf{C}$$

Analogously for colimits of type $\mathbf{J}$: when they exist, there is a functor $Colim\colon \mathbf{C}^{\mathbf{J}} \to \mathbf{C}$ computing them.

Proposition 2.2.2. *Any category of presheaves* $\mathbf{Set}^{\mathbf{C}}$ *has all limits and colimits. Given a diagram* $D\colon \mathbf{J} \to \mathbf{Set}^{\mathbf{C}}$*, for all* $c \in |\mathbf{C}|$ *we have*

$$LimD(c) = Lim(D(-)(c))$$

where $D(-)(c)\colon \mathbf{J} \to \mathbf{Set}$ *is given by* $D(x)(c) = Dx(c)$*, for* x *an object of morphism of* $\mathbf{J}$*. Similarly for colimits.*

In other words: limits and colimits in presheaf categories are computed pointwise in **Set**.

We will also use a special class of "well-behaved" presheaves, called *sheaves*. Before giving their definition, we need some additional notions (see [38],[63, Chapter 4] for details).

Definition 2.2.3 (Coverage)**.** Let $\mathbf{C}$ be a small category (i.e. a category whose objects form a set). A *coverage* on $\mathbf{C}$ is a function assigning to each object c of $\mathbf{C}$ a collection $T(c)$ of families $\{f_i\colon c \to c_i\}_{i\in I}$ of morphisms with common domain c, called *T-covering families*. These families must satisfy the following requirement:

(C-Ax) for every T-covering family $\{f_i\colon c \to c_i\}_{i\in I}$ and every morphism $g\colon c \to c'$, there exists a T-covering family $\{h_j\colon c' \to c'_j\}_{j\in J}$ such that, for each $j \in J$, there are f_i and $u\colon c_i \to c'_j$ making the following diagram commute

$$\begin{array}{ccc} c & \xrightarrow{f_i} & c_i \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle u} \\ c' & \xrightarrow[h_j]{} & c'_j \end{array} \tag{2.1}$$

Given a T-covering family $F = \{f_i\colon c \to c_i\}_{i\in I}$ and a functor $P\colon \mathbf{C} \to \mathbf{Set}$, a family $\{p_i \in P(c_i)\}_{i\in I}$ is *compatible* with P and F if, for every $g\colon c_i \to c'$ and $h\colon c_j \to c'$ such that $g \circ f_i = h \circ f_j$, we have $p_i[g]_P = p_j[h]_P$. We say that P is a *T-sheaf* if, for every T-covering family $\{f_i\colon c \to c_i\}_{i\in I}$ and compatible family $\{p_i \in P(c_i)\}$, there is a unique $p \in Pc$ such that $p[f_i]_P = p_i$, for each $i \in I$.

Definition 2.2.4 (Sheaves on a site)**.** A *site* is a pair $(\mathbf{C}, T)$ of a category $\mathbf{C}$ and a coverage T on $\mathbf{C}$. The category $\mathbf{Sh}(\mathbf{C}, T)$ of *sheaves on the site* $(\mathbf{C}, T)$ is the full subcategory of $\mathbf{Set}^{\mathbf{C}}$ whose objects are T-sheaves. We write $\mathbf{Sh}(\mathbf{C})$, and we call its objects *sheaves*, when the covering is clear from the context.

Now, consider another small category $\mathbf{D}$ and a functor $F\colon \mathbf{C} \to \mathbf{D}$. Under certain conditions, $(-) \circ F : \mathbf{Set}^{\mathbf{D}} \to \mathbf{Set}^{\mathbf{C}}$ restricts to a functor between categories of sheaves.

Definition 2.2.5 (Reflection of covers)**.** Let $(\mathbf{C}, T)$ and $(\mathbf{D}, S)$ be sites. Then a functor $F\colon \mathbf{C} \to \mathbf{D}$ *reflects covers* if, for every $c \in |\mathbf{C}|$ and every S-covering family $s \in S(Fc)$ there is a T-covering family $t \in T(c)$ such that $\{Ff \mid f \in t\}$ is contained in s.

Proposition 2.2.6. *Let $(\mathbf{C}, T)$ and $(\mathbf{D}, S)$ be sites, and consider $F\colon \mathbf{C} \to \mathbf{D}$ and $P\colon \mathbf{D} \to \mathbf{Set}$. If F reflects covers and PF is a T-sheaf then P is a S-sheaf.*

2.2.2 Locally presentable categories and accessible functors

Our theory will be developed using locally presentable categories as domains and accessible functors as constructors for syntax and semantics. Here we illustrate these notions and related results. Our main reference is [6].

Definition 2.2.7 (λ-filtered category.)**.** Let λ be an infinite regular cardinal. Then a small category $\mathbf{A}$ is *λ-filtered* if, for any category $\mathbf{B}$ such that the cardinality of $|\mathbf{B}|$ is less than λ, any diagram D in $\mathbf{A}$ of type $\mathbf{B}$ has a cocone in $\mathbf{A}$. A colimit is *λ-filtered* when it is a colimit of a functor $F\colon \mathbf{A} \to \mathbf{C}$, with $\mathbf{A}$ a λ-filtered category.

λ-filtered categories generalize the notion of directed preorders, that are sets such that every finite subset has an upper bound.

Definition 2.2.8 (Locally λ-presentable category.)**.** An object c of a category $\mathbf{C}$ is λ-presentable if the functor $\mathrm{Hom}_{\mathbf{C}}(c, -)\colon \mathbf{C} \to \mathbf{Set}$ preserves λ-filtered colimits. A category $\mathbf{C}$ is locally λ-presentable if it has all limits and there is a set of λ-presentable objects $X \subseteq |\mathbf{C}|$ such that every object is a λ-filtered colimit of objects from X.

For instance, in the category $\mathbf{Set}$, locally λ-presentable objects are precisely the finite sets with cardinality less than λ. $\mathbf{Set}$ is locally ω-presentable: every set is a ω-filtered colimit of its finite subsets and there is a set of cardinality ω that generates the whole $\mathbf{Set}$, namely the one containing one finite set of cardinality n, for all $n \in \mathbb{N}$.

Since we will work with functor categories, we need the following result.

Proposition 2.2.9. *For each locally λ-presentable category $\mathbf{A}$ and small category $\mathbf{B}$, the functor category $\mathbf{A}^{\mathbf{B}}$ is λ-presentable.*

In particular, since $\mathbf{Set}$ is ω-presentable, we have that the presheaf category $\mathbf{Set}^{\mathbf{B}}$ is ω-presentable as well.

Definition 2.2.10 (Accessible functor.)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be locally λ-presentable categories. A functor $F\colon \mathbf{A} \to \mathbf{B}$ is *λ-accessible* if it preserves λ-filtered colimits. We just say that F is accessible if it is λ-accessible for some λ.

We recall some constructions which yield accessible functors.

Proposition 2.2.11.

(i) Products, coproducts and composition of accessible functors are accessible.

(ii) Left and right adjoints between locally presentable categories are accessible ([6, Proposition 2.2.3]).

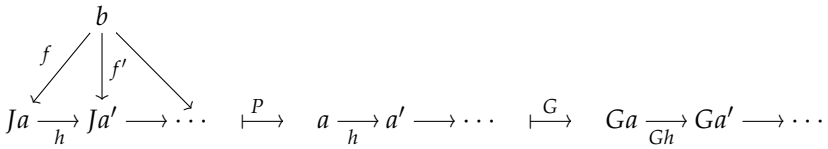
2.2.3 Kan extensions

Important operations on functor categories are *Kan extensions* [41, X]. Consider three categories $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ and a functor $J\colon \mathbf{A} \to \mathbf{B}$. There is an obvious functor $\mathscr{R}\colon \mathbf{C}^{\mathbf{B}} \to \mathbf{C}^{\mathbf{A}}$, given by precomposition with J. Under certain conditions, this functor has right and left adjoints, which we denote by $\mathscr{E}_R$ and $\mathscr{E}_L$, computing *right* and *left Kan extensions*. Their name is justified by observing that, whenever $\mathbf{A}$ is a subcategory of $\mathbf{B}$, then $\mathscr{R}$ restricts the domain of functors $\mathbf{B} \to \mathbf{C}$ to $\mathbf{A}$, and $\mathscr{E}_R, \mathscr{E}_L$ extend the domain of a functor $\mathbf{A} \to \mathbf{C}$ to the whole $\mathbf{B}$.

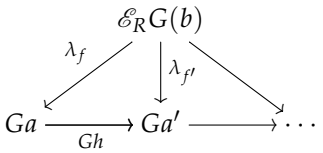
We now show the construction of $\mathscr{E}_R$ described in [41, X.3]. That of $\mathscr{E}_L$ is analogous and dual. Consider a functor $G\colon \mathbf{A} \to \mathbf{C}$. Fixed $b \in |\mathbf{B}|$, the idea is to compute $\mathscr{E}_R G(b)$ via a limit construction. Recall that the *comma category* $b \downarrow J$ is defined as follows: its objects are pairs $(f\colon b \to Ja, a)$ and its morphisms $(f\colon b \to Ja, a) \to (f'\colon b \to Ja', a')$ are morphisms $Ja \to Ja'$ in $\mathbf{B}$ that commute with f and f'. Now, consider the functor $P\colon b \downarrow J \to \mathbf{A}$, given by

$$\begin{array}{ccc} (f\colon b \to Ja, a) & \longmapsto & a \\ h\colon (f\colon b \to Ja, a) \to (f\colon b \to Ja', a') & \longmapsto & h\colon a \to a' \end{array} \tag{2.2}$$

Its composition with G gives a functor $b \downarrow J \to \mathbf{C}$, which can be depicted as



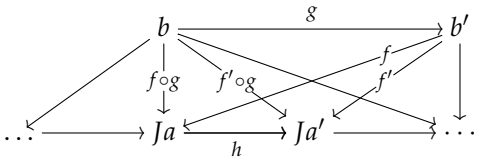
This functor can be regarded as a diagram of type $b \downarrow J$ in $\mathbf{C}$. Provided that $\mathbf{C}$ has suitable limits, the vertex of the limiting cone for GP (depicted below) gives the action of $\mathscr{E}_R G$ on b



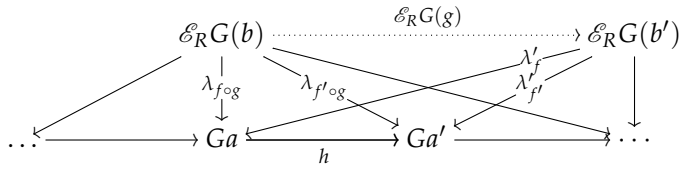
(2.3)

The action of $\mathscr{E}_R G$ on a morphism $g\colon b \to b'$ in $\mathbf{B}$ can be computed via the universal property of limits. The key observation is that g yields a correspondence between $b \downarrow J$ and $b' \downarrow J$: given $(f\colon b' \to Ja, a)$ in $b' \downarrow J$, there is an object in $b \downarrow J$ with the same codomain, namely $(fg\colon b \to Ja, a)$; consequently, for each morphism h in $b' \downarrow J$ there is a morphism h' in $b \downarrow J$ that has the same underlying morphism as h, and whose domain and codomain are

determined by those of h as described. This situation is depicted as follows.



Now, let $P' : b \downarrow J \to \mathbf{C}$ be defined similarly to (2.2). The described correspondence between $b \downarrow J$ and $b' \downarrow J$ implies that the image of GP' is contained in that of GP, so there is a cone for GP' inside every limiting cone for GP, and these cones clearly share the same vertex $\mathscr{E}_R G(b)$. Then, the value of $\mathscr{E}_R G$ at g is defined to be the unique morphism from $\mathscr{E}_R G(b)$ to the vertex of the limiting cone for GP', namely $\mathscr{E}_R G(b')$.



(2.4)

Summarizing, we have the following result.

Theorem 2.2.12 ([41, X.3, Theorem 1])**.** *Given $J : \mathbf{A} \to \mathbf{B}$, let $G : \mathbf{A} \to \mathbf{C}$ be a functor such that $GP : b \downarrow J \to \mathbf{C}$ has a limit in $\mathbf{C}$, for each $b \in |\mathbf{B}|$. Then the* right Kan extension *of G along J can be computed as follows. Its action on $b \in |\mathbf{B}|$ is given by*

$$\mathscr{E}_R G(b) := Lim(GP)$$

and its action on morphisms $g : b \to b'$ gives the unique morphism $\mathscr{E}_R G(b) \to \mathscr{E}_R G(b')$ that commutes with the involved cones.

The following corollary spells out conditions for the existence of right Kan extensions.

Corollary 2.2.13 ([41, X.3, Corollary 2])**.** *If $\mathbf{A}$ is small and $\mathbf{C}$ has all limits, any functor $G : \mathbf{A} \to \mathbf{C}$ has a right Kan extension along any $J : \mathbf{A} \to \mathbf{B}$.*

Left Kan extension, dually, are computed as pointwise colimits

$$\mathscr{E}_L G(b) = Colim((J \downarrow b) \xrightarrow{Q} \mathbf{A} \xrightarrow{G} \mathbf{C})$$

where $J \downarrow b$ is the comma category with objects of the form $(a, f : Ja \to b)$ and Q maps $(a, f : Ja \to b)$ to a.

2.2.4 Coalgebras

The *behavior* of a system can be modeled in a categorical setting through *coalgebras* [60, 4]. Given a *behavioral endofunctor* $B : \mathbf{C} \to \mathbf{C}$, describing the "shape" of a class of systems, we have a corresponding category of coalgebras.

Definition 2.2.14 (*B*-**Coalg**)**.** The category *B*-**Coalg** is defined as follows: objects are *B*-*coalgebras*, i.e. pairs (X, h) of an object $X \in |\mathbf{C}|$, called *carrier*, and a morphism $h\colon X \to BX$, called *structure map*; *B-coalgebra homomorphisms* $f\colon (X, h) \to (Y, g)$ are morphisms $f\colon X \to Y$ in $\mathbf{C}$ making the following diagram commute

$$\begin{array}{ccc} X & \xrightarrow{h} & BX \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle Bf} \\ Y & \xrightarrow[g]{} & BY \end{array}$$

For instance, let $\mathcal{P}_f\colon \mathbf{Set} \to \mathbf{Set}$ be the *finite powerset functor*, given by

$$\begin{aligned} \mathcal{P}_f X &:= \{Y \subseteq X \mid X \text{ finite}\} \\ \mathcal{P}_f h &:= \{h(Y) \mid Y \in \mathcal{P}_f X\} \qquad (h\colon X \to X') \end{aligned}$$

Given a set of labels L, a finite-branching labelled transition system $(X, \longrightarrow)$ with $\longrightarrow \subseteq X \times L \times X$, can be represented as a coalgebra (X, h) for the functor

$$B_{lts} := \mathcal{P}_f(A \times -) \ .$$

Its structure map is given by

$$h(x) \ := \ \{(x', l) \mid x \xrightarrow{l} x'\}$$

The commutative diagram for a B_{lts}-coalgebra homomorphism f amounts to saying that transitions are *preserved* and *reflected* by f, i.e.:

- whenever $x \xrightarrow{l} x'$ then $f(x) \xrightarrow{l} f(x')$ (preservation);
- i.e. whenever $f(x) \xrightarrow{l} x'$ then there is x'' such that $x \xrightarrow{l} x''$ and $x' = f(x'')$ (reflection).

Recall that a *subobject* of $X \in |\mathbf{C}|$ is an isomorphism class of monomorphisms $s\colon S \hookrightarrow X$ (two morphisms $f\colon S_1 \to X$ $g\colon S_2 \to X$ are isomorphic whenever there is an isomorphism $S_1 \to S_2$ that commutes with f and g). Using this notion, we can characterize behavioral equivalences in *B*-**Coalg**.

Definition 2.2.15 (*B*-bisimulation)**.** Given two *B*-coalgebras (X, h) and (Y, k), a *B*-*bisimulation* between them is a span $(p_1, p_2)\colon R \hookrightarrow X \times Y$ in $\mathbf{C}$ such that there is $r\colon R \to BR$ making the following diagram commute

$$\begin{array}{ccccc} X & \xleftarrow{p_1} & R & \xrightarrow{p_2} & Y \\ {\scriptstyle h}\downarrow & & {\scriptstyle r}\downarrow & & \downarrow{\scriptstyle k} \\ BX & \xleftarrow[Bp_1]{} & BR & \xrightarrow[Bp_2]{} & BY \end{array}$$

If $(Y, h) = (X, k)$ then we say that (p_1, p_2) is a *B*-bisimulation *on* (X, k). The greatest *B*-bisimulation is called *B*-bisimilarity. In the following we will leave the span implicit, using its vertex R to indicate the *B*-bisimulation.

B_{lts}-bisimulations on a B_{lts}-coalgebra (X, h) are indeed ordinary bisimulations on the equivalent LTS $(X, \longrightarrow_h)$. In fact, subobjects in **Set** are subsets, so R is a relation on X; and, given $(p, q) \in R$, for each transition $p \xrightarrow{l}_h p'$ we have one transition

$$(p, q) \xrightarrow{l}_r (p', q')$$

in $(R, \longrightarrow_r)$, due to the commutativity of the left square, but this means that there is a transition $q \xrightarrow{l}_h q'$ as well, due to the commutativity of the right square. Moreover, $(p', q') \in R$, by definition of B_{lts}.

The final coalgebra, whenever it exists, can be regarded as the universe of abstract behaviors. Roughly speaking, the unique morphism from a coalgebra to the final one assigns to each state its *abstract semantics*. A sufficient condition for the existence of final coalgebras is the following.

Theorem 2.2.16. *Every accessible endofunctor $B\colon \mathbf{C} \to \mathbf{C}$ on a locally presentable category $\mathbf{C}$ has a final coalgebra.*

The functor B_{lts} satisfies the hypothesis of Theorem 2.2.16, therefore admits a final coalgebra: its carrier contains all the finitely branching trees (quotiented by bisimilarity) whose edges are labelled by elements of L. The unique morphism from any coalgebra (X, h) associates to each state $x \in X$ the tree representing its computation.

Now we discuss the conditions under which abstract semantics and B-bisimilarity agree. Recall that a *kernel pair* of $f\colon c \to c'$ is a pair of morphisms $p_1, p_2\colon p \to c$ such that the following diagram is a pullback

$$\begin{array}{ccc} p & \xrightarrow{p_1} & c \\ {\scriptstyle p_2}\downarrow & & \downarrow{\scriptstyle f} \\ c & \xrightarrow[f]{} & c' \end{array}$$

A kernel pair of f in **Set** is indeed the kernel of f, i.e. the relation $\{(x, y) \in c \times c \mid f(x) = f(y)\}$, together with projections. Moreover, recall that a *weak pullback* is defined as a pullback, except that the uniqueness property of the mediating morphism is dropped.

Theorem 2.2.17. *If B preserves weak pullbacks, then the kernel pair of every B-coalgebra homomorphism from (X, h) is a B-bisimulation on (X, h). In particular, the kernel pair of the unique homomorphism to the final coalgebra coincides with the B-bisimilarity; in other words: the abstract semantics is fully abstract w.r.t. B-bisimulation.*

This result, instantiated to B_{lts}-coalgebras, says that two states of a coalgebra are B_{lts}-bisimilar if and only if they are are mapped to the same tree by the final morphism.

2.2.5 Coalgebras over presheaves

Whenever the behavioral endofunctor is of the form $B\colon \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{C}}$, its coalgebras and bisimulations have some properties that are worth discussing.

B-coalgebras are pairs (P, ρ) of a presheaf $P\colon \mathbf{C} \to \mathbf{Set}$ and a natural transformation $\rho\colon P \to BP$. The naturality of ρ imposes a constraint on behavior: for any $f\colon c \to c'$ the following diagram must commute

$$\begin{array}{ccc} p \in Pc & \xmapsto{\rho_c} & \rho_c(p) \in BP(c) \\ {\scriptstyle [f]_P}\downarrow & & \downarrow{\scriptstyle [f]_{BP}} \\ p[f]_P \in P(c') & \xmapsto[\rho_{c'}]{} & \rho_{c'}(p)[f]_{BP} \in BP(c') \end{array}$$

This diagram means that behavior must be preserved (top and right morphisms) and reflected (left and bottom morphisms) by the action of P on the index category morphisms.

Also bisimulations have more structure. A B-bisimulation R on (P, ρ), with $P\colon \mathbf{C} \to \mathbf{Set}$, is a presheaf such that $Rc \subseteq Pc \times Pc$, so the morphisms from R to P in the diagram of Definition 2.2.15 are projections. Moreover, since these are natural transformations, by commutativity of the following diagram we have that R is closed under the action of P on the index category morphisms.

$$\begin{array}{ccccc} p \in Pc & \xmapsfrom{(\pi_1)_c} & (p,q) \in Rc & \xmapsto{(\pi_2)_c} & q \in Pc \\ {\scriptstyle [f]_P}\downarrow & & \downarrow{\scriptstyle [f]_R = [f]_P \times [f]_P} & & \downarrow{\scriptstyle [f]_P} \\ p[f]_P \in P(c') & \xmapsfrom[(\pi_1)_{c'}]{} & (p[f]_P, q[f]_P) \in R(c') & \xmapsto[(\pi_2)_{c'}]{} & q[f] \in P(c') \end{array}$$

In order to establish a correspondence between coalgebras over presheaves and transition systems, and between coalgebraic and ordinary bisimulations, in [63] transition systems and bisimulations over indexed states are introduced. The original definition is tailored for the π-calculus, but we give a more general one. These transition systems will be used in Chapter 4 to relate coalgebraic and set-theoretic notions of behavior for our calculus.

Definition 2.2.18 (**C**-indexed labelled transition system)**.** Given a set of labels L, a **C**-*indexed labelled transition system* (**C**-ILTS) is a pair $(P, \longrightarrow)$ of a presheaf $P\colon \mathbf{C} \to \mathbf{Set}$ and a transition relation $\longrightarrow \subseteq \int P \times L \times \int P$.

Definition 2.2.19 (**C**-indexed bisimulation)**.** A **C**-*indexed bisimulation* on a **C**-ILTS $(P, \longrightarrow)$ is an indexed family of relations $\{R_c \subseteq Pc \times Pc\}_{c \in |\mathbf{C}|}$ such that, for all $(p, q) \in R_c$

(i) if $c \vdash p \xrightarrow{l} c' \vdash p'$ then there is $c' \vdash q'$ such that $c \vdash q \xrightarrow{l} c' \vdash q'$ and $(p', q') \in R_{c'}$;

(ii) for all $f\colon c \to c'$, $(p[f], q[f]) \in R_{c'}$.

2.3 The π-calculus model

This section is devoted to the π-calculus model presented in [28, 63], which is the inspiration of this thesis. We summarize the relevant results.

2.3.1 Model of resources

Finite sets of names and renamings can be modeled as the category **F**.

Definition 2.3.1 (Category **F**)**.** The category **F** has finite ordinals $n := \{1, \ldots, n\}$ as objects and functions $\sigma \colon n \to m$ as morphisms.

Each $n \in |\mathbf{F}|$ represents a set of n names, and each $\sigma \colon n \to m$ represents a renaming that only affects those names, and leaves the other ones unchanged. This category is the skeletal version (i.e. a small, equivalent subcategory such that no two different objects are isomorphic) of the category of finite sets and functions **FinSet**. Since **FinSet** has finite limits and colimits, by equivalence also **F** does.

Existence of coproducts allows modeling allocation of fresh names in **F** as

$$n \xrightarrow{old_n} n+1 \xleftarrow{new_n} 1$$

where new_n picks the fresh name from $n+1$ and old_n the old ones. This is a convenient representation, because it automatically yields a notion of capture avoiding renaming: given $\sigma \colon n \to m$, by the universal property of coproducts we have

$$\begin{array}{ccccc} n & \xrightarrow{old_n} & n+1 & \xleftarrow{new_n} & 1 \\ {\scriptstyle\sigma}\downarrow & & \downarrow{\scriptstyle\sigma+id_1} & & \downarrow{\scriptstyle id_1} \\ m & \xrightarrow[old_m]{} & m+1 & \xleftarrow[new_m]{} & 1 \end{array} \tag{2.5}$$

The morphism $\sigma + id_1$ is "capture avoiding", in the sense that it never replaces old names with fresh ones and viceversa. These constructions can be used to define an *allocation operator* $\delta \colon \mathbf{F} \to \mathbf{F}$:

$$\delta n := n+1 \qquad \delta\sigma := \sigma + id_1 \ .$$

We will also need the subcategory of **F** with only injective morphisms, denoted by **I**. The functor δ is defined on **I** as well.

2.3.2 Presheaf environment

The basic domains for processes and their semantics will be $\mathbf{Set}^{\mathbf{F}}$ and $\mathbf{Set}^{\mathbf{I}}$. We list some operators for these categories. Those in $\mathbf{Set}^{\mathbf{F}}$ also are defined in $\mathbf{Set}^{\mathbf{I}}$ as expected.

Presheaf of names. $\mathcal{N} \colon \mathbf{F} \to \mathbf{Set}$ translates n into the corresponding set of names and each morphism to the corresponding function. This is given by

$$\mathcal{N} \ := \ \mathrm{Hom}_{\mathbf{F}}(\mathbf{1}, -)$$

Explicitly, we have

$$\mathcal{N}n = \mathbf{F}[\mathbf{1}, n] \cong \{1, \ldots, n\}$$

and, given $\sigma\colon n \to m$ and $a \in \mathcal{N}n$ (i.e. $a\colon \star \to n$), we have

$$a[\sigma]_{\mathcal{N}} = \sigma \circ a$$

which is the analogous of the function application $\sigma(a)$.

Allocation operator. $\Delta\colon \mathbf{Set}^{\mathbf{F}} \to \mathbf{Set}^{\mathbf{F}}$ is the lifting of δ to presheaves, given by precomposition with δ. Explicitly

$$\Delta P(n) = P(n+1) \qquad [\sigma]_{\Delta P} = [\sigma + id_1]_P$$

In words: ΔP generates processes with one additional name than those of P for the same index, and gives renamings that preserve freshness of that name.

Finite powerset. $\mathscr{P}_f\colon \mathbf{Set}^{\mathbf{I}} \to \mathbf{Set}^{\mathbf{I}}$, acting pointwise as $\mathcal{P}_f$ (the finite powerset functor on **Set**), i.e. maps each presheaf P to the presheaf $n \mapsto \mathcal{P}_f(Pn)$. There is also a *non-empty* version $\mathscr{P}_f^+$, which discards the empty set from $\mathcal{P}_f(Pn)$.

Exponential. $P^{\mathcal{N}}\colon \mathbf{I} \to \mathbf{Set}$ given by $P^{\mathcal{N}}n = \mathbf{Set}^{\mathbf{I}}(\mathrm{Hom}_{\mathbf{I}}(n,-) \times \mathcal{N}, P)$ [41, I.6]. In [27] it is shown that a finitary description is permitted, namely

$$P^{\mathcal{N}}(n) = P(n)^n \times P(n+1) \ ,$$

because, intuitively, each natural transformation in $P^{\mathcal{N}}$ is fully characterized by its action on "known" names and on a generic new name.

Partial exponential. $P \rightharpoonup (-)$, with $P\colon \mathbf{I} \to \mathbf{Set}$, mapping $Q\colon \mathbf{I} \to \mathbf{Set}$ to the set of partial functions $\{Pn \rightharpoonup Qn\}$.

Sheaves play an important role in this model. The category $\mathbf{Sh}(\mathbf{I})$ we consider is known as the *Schanuel topos*: it consists of sheaves for the coverage T_s made of singleton families of the form $\{f\colon n \rightarrowtail m\}$ (see [63, §4.2.1] for a proof that T_s is a proper coverage). The following is a characterization result for $\mathbf{Sh}(\mathbf{I})$, given in [38, A.2.1,Example 2.1.11(h)].

Proposition 2.3.2. *Sheaves in* $\mathbf{Sh}(\mathbf{I})$ *are presheaves* $\mathbf{I} \to \mathbf{Set}$ *that preserve pullbacks.*

Given a sheaf P, $n \in |\mathbf{I}|$ and $p \in Pn$, we have a notion of *support* and *seed* of p.

Definition 2.3.3 (Support and seed)**.** We say that $i\colon m \hookrightarrow n$ *supports* p if, for all $f, g\colon n \to k$ such that $f \circ i = g \circ i$, we have $p[f] = p[g]$. The unique $p' \in Pm$ such that $p'[m \hookrightarrow n] = p$ is called *seed of* p *at* m.

Using sheaf-theoretic concepts introduced in §2.2.1, we can reformulate these definitions as follows: m supports $p \in Pn$ whenever $\{p\}$ is compatible with the singleton family $\{m \hookrightarrow$

$n\}$; the existence and uniqueness of the seed is precisely the characterizing property of T_s-sheaves. Most importantly, as shown in [30], p admits a *minimal support*, given by intersecting all its supports via pullbacks.

As for sheaves on $\mathbf{F}$, it is easy to see that T_s is also a valid coverage for $\mathbf{F}$ and the embedding $\mathbf{I} \hookrightarrow \mathbf{F}$ reflects this coverage. So, by Proposition 2.2.6, we have the following.

Proposition 2.3.4. *A presheaf P in $\mathbf{Set}^{\mathbf{F}}$ is a sheaf whenever so is $P \circ (\mathbf{I} \hookrightarrow \mathbf{F})$.*

2.3.3 Presheaf of processes

The π-calculus processes in [28] are computed as the initial algebra for a suitable signature, which incorporates α-conversion by employing Δ to model name binding (see [26] for a detailed account of categorical abstract syntax in the presence of name binding). We are not interested in the algebraic aspects, so we give a direct definition of the presheaf $\Pi\colon \mathbf{F} \to \mathbf{Set}$ of processes: its action on objects is

$$\Pi n \;:=\; \{[p]_\alpha \mid \mathrm{fn}(p) \subseteq \mathcal{N}n\} \;\;.$$

where $[p]_\alpha$ denotes a canonical representative of the α-equivalence of p; its action on a morphism $\sigma\colon n \to m$ produces the homomorphic and capture-avoiding extension of σ to processes in Πn (capture-avoidance is achieved as in (2.5)). Notice that the index of a process is always an over-approximation of its free names: this is crucial to give the semantics of name allocation.

Now, let $\Pi_{\mathbf{I}}$ be the restriction of Π to injective renamings, i.e.

$$\Pi_{\mathbf{I}} \;:=\; \Pi \circ (\mathbf{I} \hookrightarrow \mathbf{F})$$

This functor can be shown to preserve pullbacks. Then, according to Proposition 2.3.2, it is a sheaf, and from Proposition 2.3.4 we get that Π is a sheaf as well. As a consequence, we gain (minimal) supports and seeds. The notion of support, for such functors, can be restated as follows: given a process $p \in \Pi_{\mathbf{I}} n$ (=Πn), $m \hookrightarrow n$ supports p if $\mathrm{fn}(p) \subseteq m$; clearly, minimal support coincides with $\mathrm{fn}(p)$. In other words, it is always possible to find the actual free names, and the corresponding seed, of an element of $\Pi_{\mathbf{I}}$ and Π.

2.3.4 Coalgebraic semantics

The π-calculus transition system and its bisimulations cannot be modeled in $\mathbf{Set}^{\mathbf{F}}$. This is due to the intrinsic structure of coalgebras and coalgebraic bisimulations in a presheaf setting: as explained in §2.2.5, they can only model behavior and relations which are closed under the index category morphisms. In the case of $\mathbf{Set}^{\mathbf{F}}$, this means closure under all renamings. However, the π-calculus transitions system is not closed under generic renamings (Remark 2.1.4), but only under injective ones, so one can give a coalgebraic semantics in $\mathbf{Set}^{\mathbf{I}}$. We present the coalgebraic environment for the late π-calculus with early bisimulation of [28].

Definition 2.3.5 (Early behavioral functor)**.** The behavioral functor $B_e : \mathbf{Set}^{\mathbf{I}} \to \mathbf{Set}^{\mathbf{I}}$ for early bisimulation is

$$\begin{array}{rcll} B_eP & := & \mathcal{N} \rightharpoonup (\mathscr{P}_f^+ P)^{\mathcal{N}} & \text{(Input)} \\ & \times & \mathcal{N} \rightharpoonup \mathscr{P}_f^+(\mathcal{N} \times P) & \text{(Free output)} \\ & \times & \mathcal{N} \rightharpoonup \mathscr{P}_f^+(\Delta P) & \text{(Bound output)} \\ & \times & 1 \rightharpoonup \mathscr{P}_f^+ P & \text{(Silent action)} \end{array}$$

To understand this definition, consider $P\colon \mathbf{I} \to \mathbf{Set}$ and $n \in |\mathbf{I}|$. Then we have

$$\begin{array}{rcl} B_ePn & = & \{n \rightharpoonup ((\mathcal{P}_f^+ Pn)^n \times \mathcal{P}_f^+ P(n+1))\} \\ & \times & \{n \rightharpoonup \mathcal{P}_f^+(n \times Pn)\} \\ & \times & \{n \rightharpoonup \mathcal{P}_f^+ P(n+1)\} \\ & \times & \{1 \rightharpoonup \mathcal{P}_f^+ Pn\} \end{array}$$

Any structure map $P \to B_eP$ maps $p \in Pn$ to a tuple of partial functions $\langle \mathtt{in}, \mathtt{out}, \mathtt{bout}, \mathtt{tau} \rangle$: each is defined at $a \in n$ whenever p can do the corresponding action with subject a, and its image is the set of continuations after the action is performed. The interesting case is `in`: it maps a to a pair (f, X), where $f(b)$ gives the set of continuations after receiving a known name b (i.e. $b \in n$), and X those after receiving the fresh name; this models early input, because the continuation depends on the chosen input value. Notice that we wrote "*the* fresh name" because this is univocally determined by δ, according to the index of the source process. This means that all the transitions of the π-calculus transition system, differing only by the choice of the fresh communicated name, are collapsed to a unique element of X.

Definition 2.3.5 is the traditional presentation of B_e, tailored to early bisimulation on late semantics (see Definition 2.1.5), where the input channel is chosen first, and then all possible instantiations of the continuations are compared. Non-input actions are formalized in the same way for uniformity, but the following isomorphisms

$$\mathscr{P}_f(P+Q) \cong \mathscr{P}_f(P) \times \mathscr{P}_f(Q) \qquad \mathscr{P}_f(\mathcal{N} \times P) \cong \mathcal{N} \rightharpoonup \mathscr{P}_f^+(P) \qquad \mathscr{P}_f(P) \cong 1 \rightharpoonup \mathscr{P}_f^+(P)$$

allows one to give an alternative presentation:

$$B_eP := \underbrace{\mathcal{N} \rightharpoonup (\mathscr{P}_f^+ P)^{\mathcal{N}}}_{\text{Input}} \times \mathscr{P}_f(\underbrace{\mathcal{N} \times \mathcal{N} \times P}_{\text{Free output}} + \underbrace{\mathcal{N} \times \delta P}_{\text{Bound output}} + \underbrace{P}_{\text{Silent action}})$$

The behavioral endofunctor for our calculus will have a similar shape.

In [28] it shown that each functor appearing in the definition of B_e is accessible and preserves weak pullbacks, so also B_e has the same properties. Therefore, B_e-**Coalg** has a final coalgebra (Theorem 2.2.16) and B_e-bisimilarity agrees with it (Theorem 2.2.17).

The set-theoretic counterparts of B_e-coalgebras are **I**-ILTS based on the following presheaf of early labels

$$Lab_e := \underbrace{\mathcal{N} \times \mathcal{N}}_{\text{free input}} + \underbrace{\mathcal{N}}_{\text{bound input}} + \underbrace{\mathcal{N} \times \mathcal{N}}_{\text{free output}} + \underbrace{\mathcal{N}}_{\text{bound output}} + \underbrace{1}_{\text{silent}} .$$

Definition 2.3.6 (**I**-indexed labelled early transition system)**.** A **I**-*indexed labelled ground transition system* (**I**-IL$_e$TS) is a **I**-ILTS with labels in $\int Lab_e$ such that:

(i) Each $n \vdash p$ has finitely many transitions, all of the form:

- $n \vdash p \xrightarrow{ab} n \vdash p'$, $a, b \in \mathcal{N}n$ (*free input*);
- $n \vdash p \xrightarrow{a(\star)} \delta n \vdash p'$, $a \in \mathcal{N}n$ (*bound input*);
- $n \vdash p \xrightarrow{\overline{a}b} n \vdash p'$, $a, b \in \mathcal{N}n$ (*free output*);
- $n \vdash p \xrightarrow{\overline{a}(\star)} \delta n \vdash p'$, $a \in \mathcal{N}n$ (*bound output*);
- $n \vdash p \xrightarrow{\tau} n \vdash p'$ (*silent action*).

(ii) For all $\sigma \colon n \rightarrowtail m$ in **I** and $\phi \in \{\delta, Id_{\mathbf{I}}\}$:

- if $n \vdash p \xrightarrow{\mu} \phi n \vdash p'$ then $m \vdash p[\sigma] \xrightarrow{\mu[\sigma]} \phi(m) \vdash p[\phi\sigma]$ (*transitions are preserved by* σ);
- if $m \vdash p[\sigma] \xrightarrow{\mu'} \phi m \vdash p''$ then $n \vdash p \xrightarrow{\mu} \phi n \vdash p'$, with $\mu[\sigma] = \mu'$ and $p'[\phi\sigma] = p''$ (*transitions are reflected by* σ).

Proposition 2.3.7. **I**-*IL$_e$TSs are in bijection with B_e-coalgebras.*

The idea behind this equivalence is that, fixed a presheaf of states P, each kind of transition in (i) of Definition 2.3.6 "mimics" the corresponding component of B_eP. For instance, consider the function $\mathtt{in}$, modeling input in the image of a B_e-coalgebra structure map; for this we have:

- $n \vdash p \xrightarrow{ab} n \vdash p'$ whenever $(f, X) \in \mathtt{in}(a)$ and $p' \in f(b)$;
- $n \vdash p \xrightarrow{a(\star)} \delta n \vdash p'$ whenever $(f, X) \in \mathtt{in}(a)$ and $p' \in X$.

Condition (ii) of Definition 2.3.6 says that the transition relation must behave like a natural transformation. In the following we write $(P, \longrightarrow_\rho)$ for the **I**-IL$_e$TS induced by the B_e-coalgebra (P, ρ).

The notion of behavioral equivalence for **I**-IL$_e$TSs is the following.

Definition 2.3.8 (**I**-indexed early bisimulation)**.** **I**-indexed early bisimulations are **I**-indexed bisimulations on **I**-IL$_e$TSs.

There is an important aspect to highlight: **I**-indexed early bisimulations, unlike ordinary early bisimulations, do not need special side conditions to enforce freshness of communicated bound names. To see this, consider the simulation condition (i) of Definition 2.2.19 when two bound inputs are compared:

$$n \vdash p \xrightarrow{a(\star)} \delta n \vdash p' \quad \text{then} \quad n \vdash q \xrightarrow{a(\star)} \delta n \vdash q' \ ;$$

the input bound name is always the new one in δn, so it is fresh w.r.t. both p and q by construction.

B_e-bisimulations and **I**-indexed early bisimulations are related by the following statement.

Proposition 2.3.9 ([63, Proposition 3.3.13])**.** *Every B_e-bisimulation on a coalgebra (P, ρ) is equivalent to an* **I***-indexed early bisimulation on $(P, \longrightarrow_\rho)$.*

The idea is simple: any B_e-bisimulation is also a B_e-coalgebra so, by Proposition 2.3.7, it can be turned into an **I**-indexed early transition system. The states of this transition system are pairs of states of $(P, \longrightarrow_\rho)$ that can simulate each other, and each pair of simulating transitions is represented as a unique one going from the pair of sources to the pair of continuations.

The opposite correspondence is not true: as explained in [63, §3.3.3,Anomaly], if an **I**-IL$_e$TS has presheaf of states that does not preserve injections, it might be the case that some of its bisimulations cannot be turned into B_e-bisimulations. This is where sheaves $\mathbf{Sh}(\mathbf{I})$ comes into play: if, instead, the presheaf of states is a sheaf, this problem does not arise, because sheaves preserve pullbacks, therefore also injections.

Theorem 2.3.10 ([63, Theorem 4.2.5])**.** *Let (P, ρ) be a B_e-coalgebra and suppose P is a sheaf. Then B_e-bisimulations on (P, ρ) are in bijection with* **I***-indexed early bisimulations on $(P, \longrightarrow_\rho)$.*

Finally, let use instantiate this machinery to the π-calculus. We refer to [63, 3.3.3] for the construction of an **I**-IL$_e$TS $(\Pi_{\mathbf{I}}, \longrightarrow_\pi)$ for the π-calculus; this construction consists in assigning indices to processes in early π-calculus transitions.[1] Since $\Pi_{\mathbf{I}}$ is a sheaf, by Proposition 2.3.7 B_e-bisimulations on the B_e-coalgebra induced by $(\Pi_{\mathbf{I}}, \longrightarrow_\pi)$ fully correspond to **I**-indexed early bisimulation on $(\Pi_{\mathbf{I}}, \longrightarrow_\pi)$, which are clearly equivalent to ordinary early bisimulations closed under injective renamings. Among these we also have early bisimilarity, by Proposition 2.1.6.

2.3.5 Compositional semantics

Early congruence for the π-calculus is defined to be the greatest bisimulation closed under all renamings, so the most natural environment for its coalgebraic representation is $\mathbf{Set}^{\mathbf{F}}$. Fiore and Turi in [28] illustrate a way of lifting the coalgebraic machinery introduced so far to this category. The basic construct is the following adjunction

$$\mathbf{Set}^{\mathbf{F}} \underset{\mathscr{E}_R}{\overset{\mathscr{R}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Set}^{\mathbf{I}} \tag{2.6}$$

[1] Actually, the author of [63] works with presheaves on a category that is not skeletal, namely the category of finite subsets of names and injections. Skeletality slightly complicates the translation, as we will show for our calculus.

where $\mathscr{R}$ is precomposition with the embedding $\mathbf{I} \hookrightarrow \mathbf{F}$ and $\mathscr{E}_R$ is the functor giving right Kan extensions along $\mathbf{I} \hookrightarrow \mathbf{F}$ (see §2.2.3). Then, the lifting of B_e to $\mathbf{Set}^{\mathbf{F}}$ is

$$\widehat{B}_e := \mathscr{E}_R B_e \mathscr{R}$$

The category $\widehat{B}_e$-**Coalg** contains coalgebras and bisimulations that are closed under *all renamings*. The definition of $\widehat{B}_e$ ensures the existence of the final coalgebra: $\mathscr{E}_R$ and $\mathscr{R}$ are adjoints between locally presentable categories, so they are accessible and their composition with B_e is accessible as well (Proposition 2.2.11).

Moreover, since $\mathscr{E}_R$ and $\mathscr{R}$ are both right adjoints, they preserve limits, in particular (weak) pullbacks, and this property is preserved by composition. This implies that the abstract semantics induced by the final coalgebra is fully abstract w.r.t. $\widehat{B}_e$-bisimilarity.

The adjunction (2.6) gives a way of lifting B_e-coalgebras to $\widehat{B}_e$-coalgebras, and restricting $\widehat{B}_e$-coalgebras to B_e-coalgebras. Consider, in fact, the associated bijection between morphisms

$$\mathrm{Hom}_{\mathbf{Set}^{\mathbf{I}}}(\mathscr{R}P, Q) \cong \mathrm{Hom}_{\mathbf{Set}^{\mathbf{F}}}(P, \mathscr{E}_R Q) \qquad (P \in |\mathbf{Set}^{\mathbf{F}}|, Q \in |\mathbf{Set}^{\mathbf{I}}|)$$

This, instantiated with $Q = B\mathscr{R}P$, yields

$$\begin{array}{rcl} \mathrm{Hom}_{\mathbf{Set}^{\mathbf{I}}}(\mathscr{R}P, B\mathscr{R}P) & \cong & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{F}}}(P, \mathscr{E}_R B\mathscr{R}P) \\ & = & \mathrm{Hom}_{\mathbf{Set}^{\mathbf{F}}}(P, \widehat{B}_e P) \ . \end{array} \tag{2.7}$$

which can be summarized as:

Proposition 2.3.11. *$\widehat{B}_e$-coalgebras on $P \in |\mathbf{Set}^{\mathbf{F}}|$ are in bijection with B_e-coalgebras on $\mathscr{R}P$.*

This result can be instantiated to coalgebraic bisimulations, as they have associated coalgebra structure maps, which then can be extended or restricted along the adjunction.

Corollary 2.3.12. *Given a $\widehat{B}_e$-coalgebra (P, ρ), let $(\mathscr{R}P, \rho')$ be its restriction via* (2.7)*. Then a presheaf $R\colon \mathbf{F} \to \mathbf{Set}$ is a $\widehat{B}_e$-bisimulation on (P, ρ) if and only if $\mathscr{R}R$ is a B_e-bisimulation on $(\mathscr{R}P, \rho')$.*

A further step can be taken: whenever P is a sheaf, then Theorem 2.3.10 can be applied to B_e-bisimulations $\mathscr{R}R$ characterized by Corollary 2.3.12, establishing that they are equivalent to **I**-indexed early bisimulations on $(\mathscr{R}P, \longrightarrow_{\rho'})$ that are *closed under all renamings*. This result is [28, Proposition 3.1], except that P is not required to be a sheaf in the original statement, which then is true only for the direction from $\widehat{B}_e$-bisimulations to **I**-indexed early bisimulations. The sheaf property ensures that the other direction works as well.

Proposition 2.3.13 ([28, Proposition 3.1])**.** *Let (P, ρ) be a $\widehat{B}_e$-coalgebra, with P a sheaf* $\mathbf{Sh}(\mathbf{F})$*, and let $(\mathscr{R}P, \rho')$ be its restriction via* (2.7)*. Then $\widehat{B}_e$-bisimulations on (P, ρ) are in bijection with* **I***-indexed bisimulations on $(\mathscr{R}P, \longrightarrow_{\rho'})$ that are closed under all renamings.*

This result can be used to get a coalgebraic characterization of π-calculus early congruence. Take the B_e-coalgebra induced by the π-calculus **I**-IL$_e$TS $(\Pi_{\mathbf{I}}, \longrightarrow_\pi)$, and translate it to

a $\widehat{B}_e$-coalgebra using Proposition 2.3.11. Then Proposition 2.3.13 says that $\widehat{B}_e$-bisimulations on this $\widehat{B}_e$-coalgebra are equivalent to **I**-indexed early bisimulations on $(\Pi_{\mathbf{I}}, \longrightarrow_\pi)$, and thus to ordinary early bisimulations, that are closed under all renamings. In particular, the greatest such $\widehat{B}_e$-bisimulation coincides with early congruence.

Chapter 3

Network Conscious π-calculus

In this chapter we introduce the *Network Conscious π-calculus* (NCPi).

In §3.1 we motivate some design choices for our calculus through an illustrative example, inspired by Software Defined Networks. We show that having creation, provision and usage of network resources as distinct primitives is essential to model such systems.

In §3.2 we give the syntax of NCPi. We introduce site and link names and we use them as building blocks for NCPi processes. All the definitions in this section are tailored to the intuition that free names of processes should describe a graph. In fact, we take graph homomorphisms as renamings and we define well-formed processes as those without dangling links, i.e. free links with a bound endpoint. Structural congruence is given for such processes only, as α-conversion w.r.t. dangling links would be impossible.

In §3.3 we give NCPi operational semantics. We define observations as routing paths and we give SOS rules. These rules extend the π-calculus ones, but the synchronization mechanism is more complex: besides complete communications, it also derives partial paths by concatenating existing ones. Then we define *network conscious bisimilarity* as the standard bisimilarity for the NCPi transition system. We conclude by showing how to recover the π-calculus and its early semantics.

In §3.4 we give some closure properties of the transition systems, which fully reflect analogous ones of the π-calculus.

In §3.5 we deal with the congruence property: this does not hold, even considering bisimulations that are closed under all renamings, but we hint at a syntactic restriction that could be better-behaved.

Finally, in §3.6 we demonstrate the convenience of our calculus, by discussing whether and how the π-calculus can model network topologies and end-to-end routing.

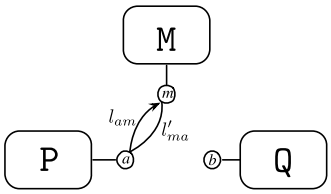


Figure 3.1: Example system.

3.1 Illustrative example

In order to have a first look at the calculus, consider the system depicted in Figure 3.1. Its aim is modeling a network where the routing structure is determined at run-time, like in SDNs. We have a network manager M, capable of creating new links and granting access to them, and two processes P and Q, which access the network through a and b respectively; they are willing to communicate, but no links exists between a and b, so P will ask M to create such link.

The formal definition is as follows

$$\mathtt{M} \stackrel{\text{def}}{=} m(x).m(y).(l_{xy})(\overline{m}l_{xy}.\mathtt{M}) \qquad \mathtt{P} \stackrel{\text{def}}{=} \overline{a}a.\overline{a}b.a(l_{ab}).(\overline{a}c.\mathtt{P}' \mid \mathtt{L}(l_{ab}))$$
$$\mathtt{L}(l_{xy}) \stackrel{\text{def}}{=} l_{xy}.\mathtt{L}(l_{xy}) \qquad \mathtt{Q} \stackrel{\text{def}}{=} b(x).\mathtt{Q}'$$

It says that M can receive two sites at M, create a new link between them and send it from M. The process P can send a and b from a, wait for a link at a and then become the parallel composition of two components: the first one can send c from a; the second one invokes the process L, whose function is activating the link l_{ab}. This activation is expressed as the *link prefix* $l_{ab}.-$ in the definition of L, and its effect is spawning a *transportation service* over l_{ab}, which can be used by the execution context to forward a datum from a to b. The link prefix expresses a single activation of the link, as input/output prefixes in the π-calculus express a single usage of their subject channel. This explains the recursive definition of L, which is intended to model a persistent connection. The process Q simply waits for a datum at b. Finally, the whole system S is the parallel composition of P, Q, M and two processes modeling a bidirectional persistent connection between a and M.

$$\mathtt{S} \stackrel{\text{def}}{=} \mathtt{P} \mid \mathtt{M} \mid \mathtt{Q} \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma})$$

Before showing some steps of computation, we briefly introduce observations by comparison with the π-calculus.

As in the π-calculus, we have observations representing inputs, output and complete communications. However, since NCPi allows for remote communications, they all include the (possibly empty) sequence of links that are traversed in the communication. For instance,

the process P can emit a at a as follows

$$\mathtt{P} \xrightarrow{\bullet;\overline{a}a} \overline{a}b.a(l_{ab}).(\overline{a}c.\mathtt{P}' \mid \mathtt{L}(l_{ab}))$$

The label $\bullet;\overline{a}a$ is a zero-length (i.e. with empty sequence of links) *output path*, which can be seen as the π-calculus action $\overline{a}a$. The symbol $\bullet$ is syntactic sugar, indicating where the path starts. In general, there may be a list of links between $\bullet$ and $\overline{a}a$, describing the path a goes through before being emitted.

Symmetrically, M can receive a at m

$$\mathtt{M} \xrightarrow{am;\bullet} (l_{ay})\overline{m}l_{ay}.\mathtt{M}$$

where $am;\bullet$ is an *input path*, analogous to the early π-calculus input action ar. Input paths always have length zero, as we only allow local receptions (this restriction will be dropped in a later chapter).

Before introducing paths denoting complete communications, we introduce *service paths*, which have no counterpart in the π-calculus. A service path has the form $a;W;b$, where W is a sequence of links. It represents a transportation service that can be used by the execution context to route a datum from a to b. For instance we have

$$\mathtt{L}(l_{am}) \xrightarrow{a;l_{am};m} \mathtt{L}(l_{am})$$

where $a;l_{am};m$ is a transportation service from a to m over l_{am}.

Finally, we have complete communication, denoted by a *complete path* $\bullet;W;\bullet$. Unlike the π-calculus τ-action, this observation is not silent, but we allow the path W of the transmitted datum to be observed; the datum itself remains unobservable. This is a precise design choice we made: as mentioned, this could be useful when modeling and analyzing routing algorithms. Another difference is that a complete path is usually produced by more than one synchronization, each one concatenating a compatible pair of paths. For instance, in order for P to communicate a to M, there must be a first synchronization between P and $\mathtt{L}(l_{ab})$, causing $\bullet;\overline{a}a$ and $a;l_{am};m$ to be concatenated

$$\mathtt{P} \mid \mathtt{L}(l_{am}) \xrightarrow{\bullet;l_{am};\overline{m}a} \dots$$

Here the continuation is the parallel composition of those shown above, and $\bullet;l_{am};\overline{m}a$ is an output path where a is emitted at m after traversing l_{am}. A complete path is produced by another, final synchronization between $\mathtt{P} \mid \mathtt{L}(l_{am})$ and M

$$\mathtt{P} \mid \mathtt{L}(l_{am}) \mid \mathtt{M} \xrightarrow{\bullet;l_{am};\bullet} \dots$$

meaning that a complete communication over l_{ab} has happened.

Now we overview the steps S can perform:

1. P **communicates to** M **the endpoints** a **and** b **of the link to be created:** it is observed as two consecutive occurrences of $\bullet;l_{ab};\bullet$. The state of the system after this interaction is

$$a(l_{xy}).(\mathtt{L}(l_{xy}) \mid \overline{a}c.\mathtt{P}') \mid (l_{ab})(\overline{m}l_{ab}.\mathtt{M}) \mid \mathtt{Q} \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma}) \ .$$

2. $\overline{m}l_{ab}.\mathtt{M}$ **communicates** l_{ab} **to** $a(l_{xy}).(\mathtt{L}(l_{xy}) \mid \overline{a}c.\mathtt{P}')$**:** we first rearrange the processes using structural congruence

$$(l_{ab})(\ a(l_{xy}).(\mathtt{L}(l_{xy}) \mid \overline{a}c.\mathtt{P}') \mid \overline{m}l_{ab}.\mathtt{M} \mid \mathtt{L}(l'_{ma})\ ) \mid \mathtt{Q} \mid \mathtt{L}(l_{am})\ .$$

Now the processes within the scope of l_{ab} can interact, and the resulting observation is $\bullet; l'_{ma}; \bullet$, with continuation

$$(l_{ab})(\mathtt{L}(l_{ab}) \mid \overline{a}c.\mathtt{P}' \mid \mathtt{M} \mid \mathtt{Q}) \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma})\ .$$

3. $\overline{a}c.\mathtt{P}'$ **communicates** c **to** $\mathtt{Q}$**:** in this case, despite l_{ab} is used for the transmission, only $\bullet;\bullet$ can be observed, because such link is restricted. This is analogous to the π-calculus τ action. The continuation is

$$(l_{ab})(\mathtt{L}(l_{ab}) \mid \mathtt{P}' \mid \mathtt{M} \mid \mathtt{Q}'[c/x]) \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma})\ .$$

3.2 Syntax

We assume an enumerable set of site names $\mathcal{S}$ (or just sites) and an enumerable set of link names $\mathcal{L}$ (or just links), equipped with two functions $s, t\colon \mathcal{L} \to \mathcal{S}$, telling source and target of each link. We denote by l_{ab} a link l such that $s(l) = a$ and $t(l) = b$. We write $\mathcal{L}_{ab}$ for the set of links of the form l_{ab} and $\mathcal{L}_a$ for the union of all $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$, for all b.

The syntax of NCPi is given in Figure 3.2: $\mathrm{n}(r)$ denotes the names in r, including a and b if r is l_{ab}. We have the usual inert process, sum and parallel composition. Prefixes can have the following forms:

- The *output prefix* $\overline{a}r$: $\overline{a}r.p$ can emit the datum r at a and continue as p.
- The *input prefix* $a(r)$: $a(r).p$ can receive at a a datum to be bound to r and becomes p.
- The *link prefix* l_{ab}: $l_{ab}.p$ can offer to the environment the service of transporting a datum from a to b through l and then continue as p.
- The τ *prefix*: $\tau.p$ can perform an internal action and continue as p.

We require that formal parameters in definitions do not have names in common, because otherwise we might have type dependencies between parameters, e.g. in $\mathtt{A}(a, l_{ab})$ one of the second parameter's endpoints depends on the first parameter.

The free names $\mathrm{fn}(p)$ of a process p are

$$\begin{aligned}
\mathrm{fn}(\mathbf{0}) &:= \varnothing & \mathrm{fn}(\tau.p) &:= \mathrm{fn}(p)\\
\mathrm{fn}(\overline{a}r.p) &:= \{a\} \cup \mathrm{n}(r) \cup \mathrm{fn}(p) & \mathrm{fn}(l_{ab}.p) &:= \{l_{ab}, a, b\} \cup \mathrm{fn}(p)\\
\mathrm{fn}(b(a).p) &:= \{b\} \cup (\mathrm{fn}(p) \setminus (\{a\} \cup \mathcal{L}_a)) & \mathrm{fn}(a(l_{bc}).p) &:= \{a, b, c\} \cup \mathrm{fn}(p) \setminus \{l_{bc}\}\\
\mathrm{fn}((a)p) &:= \mathrm{fn}(p) \setminus (\{a\} \cup \mathcal{L}_a) & \mathrm{fn}((l_{ab})p) &:= \{a, b\} \cup \mathrm{fn}(p) \setminus \{l_{ab}\}\\
\mathrm{fn}(p + q) &:= \mathrm{fn}(p \mid q) := \mathrm{fn}(p) \cup \mathrm{fn}(q) & \mathrm{fn}(\mathtt{A}(r_1, \ldots, r_n)) &:= \mathrm{n}(r_1) \cup \cdots \cup \mathrm{n}(r_n)
\end{aligned}$$

$$
\begin{aligned}
p &::= \mathbf{0} \mid \pi.p \mid p + p \mid p \,|\, p \mid (r)p \mid \mathtt{A}(r_1, r_2, \ldots, r_n) \\
\pi &::= \overline{a}r \mid a(r) \mid l_{ab} \mid \tau \\
r &::= a \mid l_{ab} \\
\mathtt{A}(r_1, r_2, \ldots, r_n) &\stackrel{\text{def}}{=} p \qquad i \neq j \implies \mathrm{n}(r_i) \cap \mathrm{n}(r_j) = \varnothing
\end{aligned}
$$

Figure 3.2: NCPi syntax.

where we must have $\mathrm{fn}(p) \subseteq \mathrm{fn}(\mathtt{A}(r_1, \ldots, r_n))$ whenever $\mathtt{A}(r_1, r_2, \ldots, r_n) \stackrel{\text{def}}{=} p$. Notice the cases $(a)p$ and $b(a).p$: a free link in p having a as endpoint is considered bound in $(a)p$ and $b(a).p$. This intuitively means that a global link cannot have private endpoints.

Now we define renamings. In the following, we shall use the usual notation $[r'_1/r_1, r'_2/r_2, \ldots, r'_n/r_n]$ to indicate the substitution mapping r_1 to r'_1, r_2 to r'_2 ... r_n to r'_n. In our case, we call *renaming* a substitution that respects the graph-structure determined by names: indeed, $\mathcal{S}$ and $\mathcal{L}$, together with s and t, can be regarded as an infinite graph. In other words, a renaming should be a *graph homomorphism*, i.e. each link should be mapped to one whose endpoints are the image through the substitution of the original link's endpoints.

Definition 3.2.1 (Renaming)**.** A renaming σ is a pair of functions $\langle \sigma_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}, \sigma_{\mathcal{L}} : \mathcal{L} \to \mathcal{L} \rangle$ such that

$$\sigma_{\mathcal{L}}(l_{ab}) = l'_{a'b'} \implies \sigma_{\mathcal{S}}(a) = a' \wedge \sigma_{\mathcal{S}}(b) = b' \ .$$

It may seem that the premise of the implication is underspecified, because $\sigma_{\mathcal{L}}$ does not say where a' and b' come from. This is not the case: a' and b' are uniquely determined by l' (namely $a' = s(l')$ and $b' = t(l')$).

In order to define the extension of renamings to processes, we need a notion of α-conversion that establishes how to avoid captures. Such notion will rely on substitutions of the form $[a'/a]$ in order to α-convert w.r.t. sites. However, such a substitution does not uniquely characterize a renaming. In fact, while surely $\overline{a}c[a'/a] = \overline{a'}c$, for $l_{ab}[a'/a]$ we only know that it must belong to $\mathcal{L}_{a'b}$. If, however, l_{ab} is bound, the choice of $l_{ab}[a'/a]$ is immaterial, provided that we remain in the same α-equivalence class. This motivates the introduction of *well-formed* processes. Informally, a process is well-formed if, in any of its subprocesses, bound links are bound explicitly, and not as a side-effect of binding a site. For instance, $a(b).l_{bc}.p$ is not well-formed because l_{bc} is implicitly bound by $a(b)$.

Definition 3.2.2 (Well-formed process)**.** A NCPi process p is *well-formed* if for every subterm q:

α-equivalence

$$(a)p \equiv (a')p[a'/a] \qquad b(a).p \equiv b(a').p[a'/a] \qquad a' \# (a)p$$
$$(l_{ab})p \equiv (l'_{ab})p[l'_{ab}/l_{ab}] \qquad c(l_{ab}).p \equiv c(l'_{ab}).p[l'_{ab}/l_{ab}] \qquad l'_{ab} \# (l_{ab})p$$

Commutative monoid laws for $\mid$ and $+$

$$p_1 \mid p_2 \equiv p_2 \mid p_1 \quad p_1 \mid (p_2 \mid p_3) \equiv (p_1 \mid p_2) \mid p_3 \quad p \mid \mathbf{0} \equiv p \quad \text{(similarly for } +)$$

Scope extension laws

$$p_1 \mid (r)p_2 \equiv (r)(p_1 \mid p_2) \qquad r \# p_1$$
$$(r)(r')p \equiv (r')(r)p \qquad r \notin \mathrm{n}(r')$$

Unfolding law

$$\mathtt{A}(r'_1, \ldots, r'_n) \equiv p[\mathrm{n}(r'_1)/\mathrm{n}(r_1), \ldots, \mathrm{n}(r'_n)/\mathrm{n}(r_n)] \quad \text{if } \mathtt{A}(r_1, \ldots, r_n) \stackrel{\text{def}}{=} p$$

Figure 3.3: Structural congruence axioms for well-formed processes.

(i) $q = (a)p'$ implies $\mathrm{fn}(q) = \mathrm{fn}(p') \setminus \{a\}$;

(ii) $q = b(a).p'$ implies $\mathrm{fn}(q) = \{b\} \cup \mathrm{fn}(p') \setminus \{a\}$;

Notice that, as a consequence of this definition, we do not need to subtract $\mathcal{L}_a$ when computing the free names of $b(a).p$ and $(a)p$, if these processes are well-formed. In the following we assume that processes are always well-formed. This allows us to define the extension of renamings to processes, which is in some sense "mutually recursive" with α-conversion.

Definition 3.2.3 (Process renamings)**.** Given a renaming σ and a well-formed process p, we denote by $p\sigma$ the result of applying σ to $\mathrm{fn}(p)$ with α-conversion of bound names to avoid captures of sites and links.

Finally, structural congruence axioms for well-formed processes are listed in Figure 3.3. Here we write $\mathrm{n}(r')/\mathrm{n}(r)$ for the substitutions $a'/a, b'/b, l'_{a'b'}/l_{ab}$ (resp. a'/a) whenever $r = l_{ab}$ and $r' = l'_{a'b'}$ (resp. $r = a$ and $r' = a'$). The interesting case is α-conversion w.r.t. a site: when α-converting $(a)p$, $[a'/a]$ is never applied to a link l_{ab}, since such link cannot be free in p by well-formedness; if it is bound, i.e. if $(l_{ab})p'$ is a subprocess of p, then we simply have inductively $((l_{ab})p')[a'/a] \equiv (l'_{a'b})p'[l'_{a'b}/l_{ab}][a'/a]$, for any $l'_{a'b}$ fresh w.r.t. p. The axioms' side conditions guarantee preservation of well-formedness.

$$\begin{aligned} \alpha &::= a;W;b \mid \bullet;W;\bullet \mid ar;\bullet \mid \bullet;W;\overline{a}r \mid \bullet;W;\overline{a}(r') \qquad r' \notin \mathrm{n}(W) \cup \{a\} \\ r, r' &::= a \mid l_{ab} \\ W &::= l_{ab} \mid W;W \mid \epsilon \end{aligned}$$

structural congruence $\equiv_\alpha$ is given by the monoidal axioms for strings, where ; is the multiplication and ϵ the identity

Figure 3.4: Syntax of paths.

3.3 Semantics

Observations of the operational semantics are routing paths, whose syntax is given in Figure 3.4. A path α can be of two general forms. It can be a *service path* $a;W;b$, representing a transportation service from a to b that employs the links listed in W and possibly other private, unobservable ones. Alternatively, it can be a string starting and/or ending with $\bullet$, which represents an actual transmission. More specifically, in this case α can be:

- an *output path*, if $\overline{a}r$ or $\overline{a}(r)$ occurs on the right, after a (possibly empty) sequence of links W: in both cases α represents r being emitted at a after traveling along W; if r is free then α is called *free output path*, otherwise α is called *bound output path* and represents an extrusion.
- an *input path*, if ar is on the left, representing the reception of r at a.
- a *complete path*, if $\bullet$ is on both sides of W, meaning that a transmission over the links in W has been completed.

Notice that input and output paths are not symmetrical: only output paths exhibit a list W of employed links. This is mainly for simplicity of presentation, and follows the intuition that a datum travels from the sender to its destination.

We call *interaction sites* of α, written $\mathrm{is}(\alpha)$ and defined in Table 3.1, those sites where the interaction with another process may happen. These correspond to subjects of the π-calculus. Table 3.1 also defines the free names $\mathrm{fn}()$, bound names $\mathrm{bn}()$ and objects $\mathrm{obj}()$. Given a list of links W and a name r, W/r removes each occurrence of r from W if $r \in \mathcal{L}$, it does nothing if $r \in \mathcal{S}$. We let α/r be α with $/r$ applied to its list of links

Definition 3.3.1 (NCPi transition system)**.** The *NCPi transition system* is the smallest transition system generated from the rules in Figure 3.5, where observations are up to $\equiv_\alpha$ and transitions are closed under $\equiv$, i.e. if $p \xrightarrow{\alpha} q$, $p \equiv p'$, $q \equiv q'$ and $\alpha' \equiv_\alpha \alpha$ then $p' \xrightarrow{\alpha'} q'$.

Now we briefly explain the rules. (OUT) and (IN) infer a zero-length path representing the beginning and the end of a transmission, respectively. We implicitly assume that r and r' in

path α	fn	bn	obj	is
$a;W;b$	$\mathrm{n}(\alpha)$	$\varnothing$	$\varnothing$	$\{a,b\}$
$\bullet;W;\bullet$	$\mathrm{n}(\alpha)$	$\varnothing$	$\varnothing$	$\varnothing$
$\bullet;W;\overline{a}r$	$\mathrm{n}(\alpha)$	$\varnothing$	$\mathrm{n}(r)$	$\{a\}$
$\bullet;W;a(r)$	$\mathrm{n}(\alpha)\setminus\{r\}$	$\{r\}$	$\mathrm{n}(r)\setminus\{r\}$	$\{a\}$
$ar;\bullet$	$\mathrm{n}(\alpha)$	$\varnothing$	$\mathrm{n}(r)$	$\{a\}$

Table 3.1: Free names, bound names, objects, input objects and interaction sites of a path α.

(IN) are of the same type, i.e. both sites or links. As in the early π-calculus, a renaming must be applied to the continuation in the free input case; if the input object is a site, then we have a substitution between sites, which can be turned into a proper renaming by well-formedness. (LINK) infers a service path made of one link. (INT) infers an internal action, represented as a complete path where everything is unobservable. (RES) computes the paths of a process with an additional restriction (r) from those of the unrestricted process, provided that r is not already bound and is not an object or an interaction site. This side condition reflects the one of the corresponding π-calculus rule, where r must not be the subject or the object of the premise's action, and its purpose is to avoid captures. In fact, suppose $b(c).p$ can perform $ba;\bullet$ and $c \in \mathrm{fn}(p)$: if $(a)b(c).p$ is allowed to perform the same path then a would be captured in its continuation $(a)p[^a/_c]$. As a result of the restriction, r becomes unobservable, so it is removed from the list of employed links in the inferred path. Notice that, if r is a site, then it is already unobservable. In fact, if r appeared as endpoint of a link in the premise' label, then this link would be free in p and $(r)p$ would not be well-formed. This justifies the fact that $/r$ has no effect when r is a site. (OPEN) treats the case, excluded by (RES), when r is the datum of a free output path: such path is turned into a bound output path, again rendering r unobservable when needed. (SUM) and (PAR) are as expected. (ROUTE), (COMP) and (COM) concatenate paths that meet at an interaction site: (ROUTE) extends an output path, provided that the transported name, whenever bound, is fresh w.r.t. the process that offers the transportation service; (COMP) composes two service paths; (COM) completes a communication.

Remark 3.3.2. For the sake of symmetry, we could have input paths that include a W component, like output paths, listing the links that a datum can traverse in order to reach the site where an input is performed. Moreover, we could have an additional inference rule, dual to (ROUTE), for adding links to such paths. However, these paths would not increase the discriminating power of the semantics. In fact, complete paths can always be derived starting with (OUT), then using (ROUTE) to attach transportation services provided by parallel processes and finally (COM) to finalize the communication.

We have that the transition system generated by these rules behaves well w.r.t. well-formedness.

$$\text{(OUT)}\quad \frac{}{\overline{a}r.p \xrightarrow{\bullet;\overline{a}r} p} \qquad \text{(IN)}\quad \frac{}{a(r).p \xrightarrow{ar';\bullet} p[r'/r]}$$

$$\text{(LINK)}\quad \frac{}{l_{ab}.p \xrightarrow{a;l_{ab};b} p} \qquad \text{(INT)}\quad \frac{}{\tau.p \xrightarrow{\bullet;\bullet} p}$$

$$\text{(SUM)}\quad \frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad \text{(OPEN)}\quad \frac{p \xrightarrow{\bullet;W;\overline{a}r} q}{(r)p \xrightarrow{\bullet;W/r;\overline{a}(r)} q}\ r{\neq}a$$

$$\text{(RES)}\quad \frac{p \xrightarrow{\alpha} q}{(r)p \xrightarrow{\alpha/r} (r)q}\ r{\notin}\mathrm{bn}(\alpha){\cup}\mathrm{obj}(\alpha){\cup}\mathrm{is}(\alpha) \qquad \text{(PAR)}\quad \frac{p_1 \xrightarrow{\alpha} q_1}{p_1 \mid p_2 \xrightarrow{\alpha} q_1 \mid p_2}\ \mathrm{bn}(\alpha)\,\#\,p_2$$

$$\text{(ROUTE)}\quad \frac{p_1 \xrightarrow{\bullet;W;\overline{a}x} q_1 \qquad p_2 \xrightarrow{a;W;b} q_2}{p_1 \mid p_2 \xrightarrow{\bullet;W;W';\overline{b}x} q_1 \mid q_2}\ \mathrm{bn}(x)\,\#\,p_2$$

$$\text{(COMP)}\quad \frac{p_1 \xrightarrow{a;W;b} q_1 \qquad p_2 \xrightarrow{b;W';c} q_2}{p_1 \mid p_2 \xrightarrow{a;W;W';c} q_1 \mid q_2}$$

$$\text{(COM)}\quad \frac{p_1 \xrightarrow{\bullet;W;\overline{a}r} q_1 \qquad p_2 \xrightarrow{ar;\bullet} q_2}{p_1 \mid p_2 \xrightarrow{\bullet;W;\bullet} q_1 \mid q_2}$$

Figure 3.5: NCPi SOS rules.

Proposition 3.3.3. *If p is well-formed and $p \xrightarrow{\alpha} q$ then q is well-formed.*

Proof. See A.1.1. □

The notion of behavioral equivalence is the following one, called *network conscious bisimulation*.

Definition 3.3.4 (Network conscious bisimulation)**.** A binary, symmetric and reflexive relation R is a *network conscious bisimulation* if $(p, q) \in R$ and $p \xrightarrow{\alpha} p'$, with $\mathrm{bn}(\alpha) \# q$, implies that there is q' such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in R$. The bisimilarity is the largest such relation and is denoted by $\sim^{NC}$.

Notice that a consequence of defining the semantics up to structural congruence is that $\equiv \subseteq \sim^{NC}$.

It is easy to see that the π-calculus is included in NCPi.

Definition 3.3.5 (Linkless NCPi)**.** We call *linkless NCPi* (NCPi$_{-\ell}$) the subcalculus of NCPi such that no links appear in processes.

Clearly, NCPi$_{-\ell}$ processes are π-calculus processes. The induced restriction on SOS rules in Figure 3.5, together with the following encoding of π-calculus actions

$$\begin{aligned} \overline{a}x &\longmapsto \bullet; \overline{a}x \quad x \in \{b, (b)\} \\ ab &\longmapsto ab; \bullet \\ \tau &\longmapsto \bullet; \bullet \end{aligned}$$

give the following proposition. The proof is immediate: rules in Figure 3.5 boil down to the early π-calculus ones (see Figure 2.2) under the given encoding.

Proposition 3.3.6. *NCPi$_{-\ell}$ transitions are in bijective correspondence with π-calculus early transitions.*

3.4 Closure properties

Here we list some properties of the transition system and its bisimulations, namely closure under some classes of renamings and contexts. Their proofs are standard.

We say that a renaming σ is *injective* if so are $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{L}}$. We have that the transition system is closed under injective renamings.

Proposition 3.4.1. *Let p be a process and σ an injective renaming, then:*

(i) if $p \xrightarrow{\alpha} p'$ then $p\sigma \xrightarrow{\alpha\sigma} p'\sigma$ (transitions are preserved by σ);

(ii) if $p\sigma \xrightarrow{\alpha} p'$ then there is $p \xrightarrow{\alpha'} p''$ such that $\alpha'\sigma \equiv_\alpha \alpha$ and $p''\sigma \equiv p'$ (transition are reflected by σ).

Remark 3.4.2. Transitions are *not* reflected by generic renamings. In fact, consider the process

$$p = \overline{a}c.\mathbf{0} \mid l_{ab}.\mathbf{0} \mid b'(x).\mathbf{0}$$

and the renaming σ that maps b' to b and acts as the identity elsewhere. Then we have

$$p\sigma \xrightarrow{\bullet;l_{ab};\bullet} \mathbf{0}$$

but there is no α' such that $p \xrightarrow{\alpha'} \mathbf{0}$ and $\alpha'\sigma \equiv_\alpha \bullet; l_{ab}; \bullet$.

Proposition 3.4.3. *$\sim^{NC}$ is closed under injective renamings, i.e. $p \sim^{NC} q$ implies $p\sigma \sim^{NC} q\sigma$, for all injective σ.*

3.5 Congruence property

Now we investigate closure under operators of the syntax. As expected, closure under input prefix does not hold. Unfortunately, also the parallel composition operator is problematic.

Example 3.5.1. Consider the following processes

$$p = l_{ab} \mid l'_{cd} \qquad q = l_{ab}.l'_{cd} + l'_{cd}.l_{ab} \ ,$$

the latter being the interleaving unrolling of the former. We have $p \sim^{NC} q$, but if we put l''_{bc} in parallel to both processes

$$p \mid l''_{bc} \xrightarrow{a;l_{ab};l''_{bc};l'_{cd};d} \mathbf{0} \qquad q \mid l''_{bc} \not\xrightarrow{a;l_{ab};l''_{bc};l'_{cd};d}$$

so $p \mid l''_{bc} \not\sim^{NC} q \mid l''_{bc}$.

So the closure result is the following.

Theorem 3.5.2. *$\sim^{NC}$ is closed under all operators except input prefix and parallel composition.*

Proof. See §A.1.2. □

However, closure under renamings does help. Consider the following relation.

Definition 3.5.3. Let $\approx^{NC}$ be the greatest bisimulation closed under all renamings, namely

$$\approx^{NC} := \ \{(p,q) \mid \forall \sigma : p\sigma \sim^{NC} q\sigma\} \ .$$

Proposition 3.5.4. *$\approx^{NC}$ is also closed under input prefix.*

Proof. See §A.1.3. □

Moreover, $\approx^{NC}$ is able to distinguish the processes p and q of Example 3.5.1. In fact, if we take any renaming σ that maps b to c, l_{ab} to $\tilde{l}_{ac}$ and l'_{cd} to $\tilde{l}'_{cd}$, we have

$$p\sigma \xrightarrow{a;\tilde{l}_{ac};\tilde{l}'_{cd};d} \mathbf{0} \quad \text{but} \quad q\sigma \not\xrightarrow{a;\tilde{l}_{ac};\tilde{l}'_{cd};d}$$

The discriminative power of $\approx^{NC}$ is still not enough, as the following example shows.

Example 3.5.5. For the following processes

$$p' = \overline{a}r \mid l_{ab} \mid c(x)$$
$$q' = \overline{a}r \mid (\ (l_{ab} \mid c(x)) + (l_{ab}.c(x) + c(x).l_{ab})\ )$$

we have $p' \approx^{NC} q'$, but again $p' \mid l'_{bc} \xrightarrow{\bullet; l_{ab}; l'_{bc}; \bullet} \mathbf{0}$, which $q' \mid l'_{bc}$ cannot simulate. This is because $l_{ab} \mid c(x)$ is within a sum in q', so it is forced to interact in an interleaving manner with $\overline{a}r$. This example suggests that some combinations of parallel and sum should be forbidden.

Finding a suitable syntactic restriction for which $\approx^{NC}$ is a congruence remains an open problem. We conjecture that a good candidate is NCPi with *guarded sums*, where prefixes and sums are replaced with

$$\sum_{i=1,\ldots,n} \pi_i.p_i \qquad (n \in \mathbb{N})\ .$$

In fact, we only have processes that are structurally congruent to:

$$(R)(\ \sum_{i=1,\ldots,n_1} \pi_i^1.p_i^1 \mid \ldots \mid \sum_{i=1,\ldots,n_k} \pi_i^k.p_i^k \quad)$$

where R is a sequence of restrictions; q' in Example 3.5.5 is not of this form, so it is ruled out. This supports the validity of the following conjecture.

Conjecture 3.5.6. *$\approx^{NC}$ is a congruence for NCPi with guarded sums.*

3.6 NCPi vs the π-calculus

Here we want to discuss how convenient is NCPi with respect to the π-calculus, when one wants to model network topologies and end-to-end routing. We can imagine to represent a link l_{ab} as a π-calculus link process $L(l, a, b)$ whose core is a forwarding process of the form $a(x).\overline{b}x$. A network would be represented as the parallel composition of link processes. In the following we give evidence that this is not convenient.

First of all, since we want to observe links used in a communication, $L(l, a, b)$ should explicitly exhibit l, a and b each time the link is used, for instance through output actions involving these names, before forwarding the datum. This is because the forwarding gives rise to a completely silent action. The same behavior in NCPi is achieved in a simpler way, through a dedicated prefix.

Even if we admitted link processes, we would not be able to observe whole routing paths in a single π-calculus transition. This has consequence on bisimulation: in the π-calculus we compare processes according to their possible forwarding alternatives, in NCPi according to traces made of such forwardings. We could define an ad-hoc π-calculus bisimulation that considers sequences of transitions, but the theory would be quite complex.

Simulating link passing in the π-calculus would be cumbersome: it requires a higher-order calculus. Even if only the link name l is passed, instead of the whole process $L(l, a, b)$,

there would be a considerable overhead in keeping track of which names denote links. This would possibly require a type system.

Finally, in the π-calculus we would completely lose the graph structure at the level of names, so a renaming could have an inconsistent behavior, e.g. it could substitute link names for site names. In order to recover such structure, and to force renamings to respect it, we could introduce a suitable type system, but its complexity would be comparable, if not greater, to that of an ad-hoc extension like ours.

Chapter 4

Coalgebraic semantics of NCPi

Coalgebras are convenient operational models, as they have been studied in full generality, regardless of the specific structure of systems, and algorithms and logics have been developed for them. Therefore, once one characterizes the operational semantics and behavioral equivalence of a language in coalgebraic terms, a plethora of results and techniques become available, e.g. minimization algorithms.

The goal of this chapter is the coalgebraic treatment of NCPi. The general results are:

1. Validation of the presheaf-based coalgebraic approach of [28] by applying it to a case which is substantially more complex than the π-calculus;
2. Integration of models with such additional expressivity, namely presheaf-based coalgebras, HD-automata and saturated semantics.

In §4.1 we give an overview of the steps we will follow.

In §4.2 we introduce our categorical environment. The most innovative part is the categorical characterization of networks and their allocation operators. In particular, we are able to model allocation of links in a way that is independent from the structure of the involved network, thus functorial: we allocate links between every pair of nodes. This is possible because our networks are multigraphs, so we can always add a new edge between two nodes; plain graphs would be harder to handle. Then we introduce presheaves and sheaves over networks, and we give some operators that will be used to construct coalgebras. We prove that all these categories and operators have the "right" properties that allow them to fit in the framework of [28].

In §4.3 we introduce the coalgebraic environment. We define a behavioral functor for NCPi which uses allocation operators to model input and output paths of fresh data. This requires some ingenuity: while our allocation operator for links creates all possible fresh links, the behavioral functor selects a specific one; this allows us to recover the correspondence with the labelled semantics. This functor is well-behaved, in the sense that its category of coalgebras has the final object, and coalgebraic bisimulations are fully abstract

with respect to the final semantics. Then we construct the coalgebra for NCPi and we show that processes indeed form a sheaf. This is necessary for establishing the equivalence of ordinary bisimulations and coalgebraic ones. In order to prove this equivalence, we adopt an intermediate representation of coalgebras as transition systems with indexed states, which enables us to use results from [63].

In §4.4 we apply a result of [17] to prove the existence of HD-automata for NCPi. In other words, we show that is possible to do finite-state verification of our calculus.

In §4.5 we construct a new NCPi coalgebra in a category where behavior is closed under renamings. This is done through Kan extensions, as in [28], but we give an explicit characterization of Kan extensions as saturation, developing the results of [8]. Other differences with [28] and [8] are: more complex categories, additional results (e.g. existence of the final coalgebra) and the fact that we consider coalgebras over sheaves. Again, sheaves allow us to relate coalgebraic and ordinary bisimulations closed under renamings.

4.1 The approach

We begin by summarizing the method for constructing coalgebraic models of nominal calculi. The high level steps are:

(i) selecting a category $\mathbf{C}$ of indices which represents resources and their operations, together with endofunctors $\delta\colon \mathbf{C} \to \mathbf{C}$ that model resource generation;

(ii) modeling processes and renamings as a syntactic presheaf in $\mathbf{Set}^{\mathbf{C}}$;

(iii) modeling the transition system as a coalgebra over the syntactic presheaf, for a behavioral endofunctor where each δ is used to represent a way of allocating resources along transitions.

Recall from §2.3.1 that $\mathbf{C}$, in the case of the π-calculus, is the category $\mathbf{F}$ of finite ordinals and functions, equipped with an endofunctor $\delta\colon \mathbf{F} \to \mathbf{F}$ that adds a fresh name to its argument. The syntactic presheaf, described in §2.3.3, maps each set of names to processes where (at most) those names occur free. However, there is a difficulty in step (iii): while syntax can be modeled in $\mathbf{Set}^{\mathbf{F}}$, the transition system and its bisimilarity cannot, because they do not fulfill the requirement of closure under the index category morphisms. The solution is splitting step (iii) in two substeps:

(iii.a) identify a subcategory $\mathbf{C}'$ of $\mathbf{C}$ such that the transition system is closed under its morphisms, and construct a coalgebra in $\mathbf{Set}^{\mathbf{C}'}$ by suitably restricting the syntactic presheaf;

(iii.b) recover a coalgebra in $\mathbf{Set}^{\mathbf{C}}$ via *right Kan extension* (see §2.2.3 and [41]) along the embedding $\mathbf{C}' \hookrightarrow \mathbf{C}$.

The category $\mathbf{C}'$, for the π-calculus, is $\mathbf{I}$, the subcategory of $\mathbf{F}$ with only injections. As shown in §2.3.4, a faithful coalgebraic representation of the π-calculus transition system and of observational equivalence is then feasible, as these are known to be closed under injective renamings. Step (iii.b) is illustrated in §2.3.5: it consists in lifting the coalgebra of (iii.a) to $\mathbf{Set}^{\mathbf{F}}$ along the adjunction associated to right Kan extensions. In the resulting category of coalgebras the greatest bisimulation characterizes observational (early/late) congruence, because behavior is always closed under all renamings.

We will consider all these steps. As mentioned in §3.5, the last step will not produce observational congruence for full NCPi, as closure under parallel operator remains to be fixed. However, closure under all renamings can possibly give closure under parallel composition for NCPi with guarded sums (Conjecture 3.5.6). We will also consider the construction of a finite-state representation of our coalgebraic semantics, in the form of a HD-automaton. Such coalgebraic semantics, in fact, will have an infinite number of states, due to lack of deallocation along transitions. For the π-calculus, this has been done by exploiting the equivalence between coalgebras on $\mathbf{Sh}(\mathbf{I})$ and HD-automata [30, 63]. A recent generalization [17] characterizes a spectrum of presheaf categories that admit HD-automata. We will employ this result to show the existence of a HD-automaton corresponding to the NCPi coalgebra with ordinary bisimulation.

4.2 Categorical environment

The development of §2.3.1 suggests some guidelines for the formal modeling of resources: one should have a small category made of finite collections of resources and their morphisms, with enough limits and colimits. In fact, for the π-calculus, finite sets of channels and renamings are modeled as $\mathbf{F}$, a skeleton of the category of finite sets and functions **FinSet**, from which $\mathbf{F}$ inherits universal structures. Following these guidelines, we model resources of NCPi processes, namely communication networks, as finite, directed multigraphs, and we define a category made of this kind of graphs and their homomorphisms. We adopt the presentation of such graphs as functors from the category $\rightrightarrows$ with two objects, representing vertices and edges, and two parallel morphisms, representing source and target operations, to the category **FinSet**. However, we just take a skeletal category of $\mathbf{FinSet}^{\rightrightarrows}$.

Definition 4.2.1 (Category $\mathbf{G}$)**.** We denote by $\mathbf{G}$ the skeletal category of $\mathbf{FinSet}^{\rightrightarrows}$.

We don't give an explicit construction for $\mathbf{G}$: all choices are consistent, since they are all isomorphic. This is why we refer to $\mathbf{G}$ as "the" skeletal category.

Concretely, we can regard each $g \in |\mathbf{G}|$ as a tuple (v_g, e_g, s_g, t_g), where v_g, e_g are the sets of vertices and edges of g, and $s_g, t_g \colon e_g \to v_g$ tell the source $s_g(e)$ and target $t_g(e)$ of each $e \in e_g$. A morphism $\sigma \colon g \to g'$ is a natural transformation, i.e. a pair of functions (σ_v, σ_e) that commute with the source and target functions of g and g', which is exactly the definition of graph homomorphism.

We state some properties of $\mathbf{G}$ that will be important in the following. Recall that monomorphisms are *stable under pushouts* if, given any pushout

$$\begin{array}{ccc} g_1 & \xrightarrow{\sigma_1} & g_2 \\ {\scriptstyle \sigma_2}\downarrow & \lrcorner & \downarrow{\scriptstyle \sigma_3} \\ g_3 & \xrightarrow[\sigma_4]{} & g_4 \end{array}$$

σ_1 (resp. σ_2) monic implies σ_4 (resp. σ_3) monic.

Proposition 4.2.2. *The category* $\mathbf{G}$ *is small, has finite colimits and pullbacks, and monos are stable under pushouts in it.*

Proof. See §A.2.1. □

$\mathbf{FinSet}^{\rightrightarrows}$ is locally small, but not small: this is why we consider a skeletal version of it. Some notation: we write $[n]$ for the discrete graph with n vertices, and k_n for the graph with n vertices and with one edge between every (ordered) pair of vertices.

Thanks to Proposition 4.2.2, we can exploit colimits to implement two allocation operators, one for vertices and one for edges. The former is very similar to the allocation endofunctor $\delta\colon \mathbf{F} \to \mathbf{F}$ presented in §2.3.1 (in fact, the action on discrete graphs is the same). The latter is more involved, and has no counterpart in the π-calculus model.

Allocation of vertices. Given $g \in |\mathbf{G}|$, we can express the allocation of a fresh, disconnected vertex $\star$ as a coproduct

$$[1] \xrightarrow{new^{\bullet}_g} g + [1] \xleftarrow{old^{\bullet}_g} g$$

This induces the endofunctor $\delta^{\bullet}\colon \mathbf{G} \to \mathbf{G}$ given by

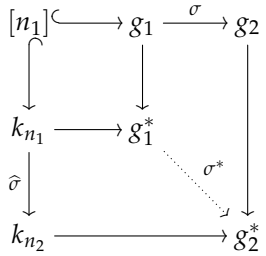
$$\delta^{\bullet} g := g + [1] \qquad \delta^{\bullet}\sigma := \sigma + id_{[1]} \ .$$

Allocation of edges. Given $g \in |\mathbf{G}|$ with n vertices, we can add a new edge $\star_{ij}$ between each ordered pair of vertices i and j through a pushout

$$\begin{array}{ccc} [n] & \hookrightarrow & g \\ \downarrow & \lrcorner & \downarrow{\scriptstyle old^{\bullet\to\bullet}_g} \\ k_n & \xrightarrow[new^{\bullet\to\bullet}_g]{} & g^* \end{array}$$

that makes the disjoint union of the items of k_n and g, and then identifies the vertices that are image of the same vertex in $[n]$ through the embeddings. Given g_1 and g_2 in $\mathbf{G}$, with n_1 and n_2 vertices respectively, every $\sigma\colon g_1 \to g_2$ in $\mathbf{G}$ can be canonically extended to a morphism

$\sigma^* : g_1^* \to g_2^*$ via the universal property of pushouts as follows



where $\widehat{\sigma}$ is the (unique) morphism between k_{n_1} and k_{n_2} that acts on vertices like σ (the action on edges is obvious).

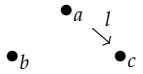
Proposition 4.2.3. *The following maps*

$$\delta^{\bullet\to\bullet} g \;:=\; g^* \qquad \delta^{\bullet\to\bullet}\sigma \;:=\; \sigma^*$$

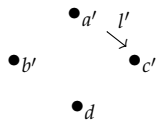
define a functor $\delta^{\bullet\to\bullet} : \mathbf{G} \to \mathbf{G}$.

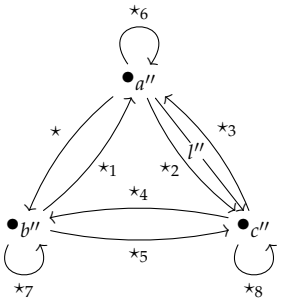
Proof. See §A.2.2. □

Example 4.2.4. Consider the following graph

$g \quad :=$ 

$\delta^\bullet g$ and $\delta^{\bullet\to\bullet} g$ are (isomorphic to)





where a', b', c' and l' (resp. a'', b'', c'', l'') are the images of a, b, c and l via $old^\bullet$ (resp. $old^{\bullet\to\bullet}$), d is the image of $\bullet$ via $new^\bullet$ and $\star, \star_1, \ldots, \star_8$ are the images of the edges in k_3 via $new^{\bullet\to\bullet}$.

We denote by $\mathbf{G}_\mathrm{I}$ the subcategory of $\mathbf{G}$ with only monos. We remark that $\mathbf{G}_\mathrm{I}$ lacks pushouts, but we can compute them in $\mathbf{G}$, since monos are stable under pushouts in $\mathbf{G}$. Consequently, $\delta^\bullet$ and $\delta^{\bullet\to\bullet}$ are well-defined also in $\mathbf{G}_\mathrm{I}$.

Now we look at the category $\mathbf{Set}^\mathbf{G}$ of presheaves on graphs. Since $\mathbf{G}$ is small, then $\mathbf{Set}^\mathbf{G}$ is locally presentable (Proposition 2.2.9), which allows endofunctors on this category to be accessible. We introduce some operators on $\mathbf{G}$ and $\mathbf{Set}^\mathbf{G}$. Most of them have a π-calculus counterpart in §2.3.2, which inspired their definition.

Name functors. $\mathcal{S}, \mathcal{L} \colon \mathbf{G} \to \mathbf{Set}$ giving, for each $g \in |\mathbf{G}|$, the set of sites and links corresponding to the vertices and edges of g. Formally, let $\bullet$ be the graph with one vertex and no edges, and $\bullet \to \bullet$ the graph with two vertices and one edge between them. We define

$$\mathcal{S} := \mathrm{Hom}_{\mathbf{G}}(\bullet, -) \qquad \mathcal{L} := \mathrm{Hom}_{\mathbf{G}}(\bullet \to \bullet, -) \qquad \mathcal{N} := \mathcal{S} + \mathcal{L}$$

Explicitly, $\mathcal{S}$ sends $g \in |\mathbf{G}|$ to $\mathbf{G}[\bullet, g]$, which is isomorphic to the set of vertices of g, and sends $\sigma \colon g \to g'$ to the function $\lambda s \in \mathrm{Hom}_{\mathbf{G}}(\bullet, g).\sigma \circ s$, which renames the site s according to σ; similarly for $\mathcal{L}$. In order to keep the same notation for names introduced in §3.2, we use a to indicate the homomorphism $\bullet \to g$ in $\mathcal{S}g$ that maps $\bullet$ to $a \in v_g$; and l_{ab} to indicate the homomorphism $(\bullet \to \bullet) \to g$ in $\mathcal{L}g$ that maps the edge in the domain to $l \in e_g$, and consequently its endpoints to $a = s_g(l)$ and $b = t_g(l)$.

Countable powerset. $\mathscr{P}_c \colon \mathbf{Set}^{\mathbf{G}} \to \mathbf{Set}^{\mathbf{G}}$, acting pointwise as the functor $\mathcal{P}_c \colon \mathbf{Set} \to \mathbf{Set}$ given by

$$\begin{aligned} \mathcal{P}_c X &:= \{Y \subseteq X \mid Y \text{ is countable}\} \\ \mathcal{P}_c f &:= \{f(Y) \mid Y \in \mathcal{P}_c X\} \qquad\qquad (f \colon X \to X') \end{aligned}$$

Allocation functors. $\Delta^{\bullet}, \Delta^{\bullet\to\bullet} \colon \mathbf{Set}^{\mathbf{G}} \to \mathbf{Set}^{\mathbf{G}}$, given by precomposition with $\delta^{\bullet}$ and $\delta^{\bullet\to\bullet}$, respectively. Namely

$$\Delta^{\bullet} P(g) := g + [1] \qquad \Delta^{\bullet\to\bullet} P(g) := g^* \ .$$

Proposition 4.2.5. *$\lambda P \in |\mathbf{Set}^{\mathbf{G}}|.\mathcal{S}, \lambda P \in |\mathbf{Set}^{\mathbf{G}}|.\mathcal{L}$ (i.e. $\mathcal{S}$ and $\mathcal{L}$ seen as constant endofunctors on $\mathbf{Set}^{\mathbf{G}}$), $\mathscr{P}_c$, $\Delta^{\bullet}$ and $\Delta^{\bullet\to\bullet}$ are accessible and preserve weak pullbacks.*

Proof. See §A.2.3. □

Finally, we are interested in characterizing sheaves on $\mathbf{G}_{\mathrm{I}}$ and $\mathbf{G}$. Taking inspiration from the definition of $\mathbf{Sh}(\mathbf{I})$ (see §2.3.2), we can define $\mathbf{Sh}(\mathbf{G}_{\mathrm{I}})$ as the category of sheaves for the coverage made of singleton families $\{f \colon g \rightarrowtail g'\}$. This is a proper coverage. In fact, given f in such a coverage family, we can compute the morphisms required by **C-Ax** of Definition 2.2.3 by letting diagram (2.1) be a pushout: by Proposition 4.2.2, its leg parallel to f is monic, then forms a coverage family. Since $\mathbf{Sh}(\mathbf{G}_{\mathrm{I}})$ and $\mathbf{Sh}(\mathbf{I})$ use very similar coverages, [38, A.2.1,Example 2.1.11(h)] can be easily adapted to $\mathbf{Sh}(\mathbf{G}_{\mathrm{I}})$, and gives us the following result.

Proposition 4.2.6. *Sheaves in $\mathbf{Sh}(\mathbf{G}_I)$ are presheaves $\mathbf{G}_I \to \mathbf{Set}$ that preserve pullbacks.*

Now, consider the embedding $J \colon \mathbf{G}_{\mathrm{I}} \hookrightarrow \mathbf{G}$ and the functor $\mathscr{R} \colon \mathbf{Set}^{\mathbf{G}} \to \mathbf{Set}^{\mathbf{G}_{\mathrm{I}}}$, given by precomposition with J. Since the singleton coverage is also valid for $\mathbf{G}$, and J trivially reflects covers, we can restate Proposition 2.2.6 in this context as follows.

Proposition 4.2.7. *Given $P \colon \mathbf{G} \to \mathbf{Set}$, if $\mathscr{R}P$ is a sheaf in $\mathbf{Sh}(\mathbf{G}_I)$ then P is a sheaf in $\mathbf{Sh}(\mathbf{G})$.*

Remark 4.2.8. Our edge allocation operator $\delta^{\bullet\to\bullet}$ may seem inefficient: it generates fresh edges between every pair of vertices, but only one of them will appear in the continuation after an allocating transition. However, this operation is uniform on all graphs, so it is functorial. Having a functor on $\mathbf{G}$ allows us to lift it to presheaves in a way that ensures the existence of both left and right adjoint (giving Kan extensions along $\delta^{\bullet\to\bullet}$) for the lifted functor, and then preservation of both limits and colimits, which is essential for coalgebras employing such functor. Generation of unused resources is not really an issue: as we will see later, NCPi processes form a sheaf, so it is always possible to recover their minimal support, i.e. the minimal amount of resources they use.

We could give an alternative definition, that operates directly on presheaves:

$$\Delta^{\bullet\to\bullet}P(g) \;:=\; \sum_{\ell\colon [2]\to g} P(g_\ell)$$

where ℓ picks a pair of vertices in g and g_ℓ is g with a new edge between those vertices. On the one hand, this avoids wasting resources, because it generates processes indexed by graphs with just one additional edge. On the other hand, this definition has the conceptual disadvantage of using "implementation details" of resources at the level of syntax and semantics. Moreover, it does not have clear properties.

4.3 Coalgebraic semantics

Our aim now is to construct a coalgebra that models the NCPi transition system. Its carrier will be a suitable presheaf modeling processes and renamings. However, since transitions are not reflected by generic renamings (Remark 3.4.2), but only by injective ones (Proposition 3.4.1), according to step (iii.a) of §4.1 we first give a semantics in $\mathbf{Set}^{\mathbf{G}_\mathrm{I}}$.

Definition 4.3.1 (Behavioral endofunctor)**.** Let $\mathcal{L}^\star = \sum_{i\in\omega} \mathcal{L}^i$. The behavioral endofunctor $B_n\colon \mathbf{Set}^{\mathbf{G}_\mathrm{I}} \to \mathbf{Set}^{\mathbf{G}_\mathrm{I}}$ is

$$\begin{aligned}
B_nP = \mathscr{P}_c(&\mathcal{S}\times\mathcal{L}^\star\times\mathcal{S}\times P && \text{(Service Path)}\\
+\;&\mathcal{L}^\star\times P && \text{(Complete Path)}\\
+\;&\mathcal{S}\times\mathcal{N}\times P && \text{(Known Name Input Path)}\\
+\;&\mathcal{S}\times\Delta^\bullet P && \text{(Fresh Site Input Path)}\\
+\;&\mathcal{S}\times\mathcal{S}\times\mathcal{S}\times\Delta^{\bullet\to\bullet}P && \text{(Fresh Link Input Path)}\\
+\;&\mathcal{L}^\star\times\mathcal{S}\times\mathcal{N}\times P && \text{(Free Output Path)}\\
+\;&\mathcal{L}^\star\times\mathcal{S}\times\Delta^\bullet P && \text{(Bound Site Output Path)}\\
+\;&\mathcal{L}^\star\times\mathcal{S}\times\mathcal{S}\times\mathcal{S}\times\Delta^{\bullet\to\bullet}P) && \text{(Bound Link Output Path)}
\end{aligned}$$

To understand this definition, consider a B_n-coalgebra (P,ρ). Given $g\in\mathbf{G}_\mathrm{I}$ and $p\in Pg$, $\rho_g(p)$ is a countable set of tuples. These tuples can be seen as pairs (α,p') of a path α and

of a continuation from p after observing α, both built using the names corresponding to the items of g and possibly some fresh ones.

We use the countable powerset, instead of the finite one, because p might have recursive subprocesses that generate a countable number of looping paths. This does not affect the formal properties of B_n-coalgebras. A finite-branching fragment of the calculus, more suitable for model checking, could be obtained by preventing SOS rules from concatenating paths that go through the same site. In fact, since our graphs are finite, there is only a finite number of paths that touch all different sites.

Notice the bound output cases: the continuation is drawn from $\Delta^{\bullet}P(g)$ or $\Delta^{\bullet\to\bullet}P(g)$, i.e. its index is $\delta^{\bullet}g$ or $\delta^{\bullet\to\bullet}g$; the extruded name, which corresponds to the new vertex or one of the new edges added to g by these functors, does not need to not appear in α, because its identity is univocal. In the bound link output case the endpoints of the extruded link must be included in α, in order to allow processes that extrude links with different endpoints through the same path to be distinguished. The W component in Figure 3.4 is modeled through the functor $\mathcal{L}^*$, which returns the set of finite strings on the alphabet $\mathcal{L}g$.

Example 4.3.2. To show how our formalization of bound link output works, let us assume an informal notion of presheaf of processes, indexing processes with graphs that include the graph of their free names (the formal definition will be given later). Consider the process $a(y) \mid (l_{ab})\overline{a}l_{ab}.l_{ab}$. It can be indexed by the graph g made of two nodes a and b. The NCPi semantics yields transitions

$$a(y) \mid (l_{ab})\overline{a}l_{ab}.l_{ab} \xrightarrow{\bullet;\overline{a}(l'_{ab})} a(y) \mid l'_{ab} \ ,$$

for all possible l'_{ab}. However, Definition 4.3.1 states that the index of the continuation must be

$$\delta^{\bullet\to\bullet}g \quad = \quad \overset{\star_{aa}}{\circlearrowleft}\ \bullet_a \xrightarrow{\star'_{ab}} \bullet_b \ \overset{\star''_{bb}}{\circlearrowleft}$$

so we only know that some fresh links have been generated along the transition. More information is given by the bound output path constructed by B_n: the last two occurrences of $\mathcal{S}$ in its definition indicate the endpoints of the fresh link. In this case B_n selects the following transition only:

$$g \vdash a(y) \mid (l_{ab})\overline{a}l_{ab}.l_{ab} \xrightarrow{\bullet;\overline{a}(\star'_{ab})} \delta^{\bullet\to\bullet}g \vdash a(y) \mid \star'_{ab} \ .$$

We remark that the name of the fresh link does not actually appear in the path generated by B_n, because it is unique.

Input transitions are modeled similarly to free and bound output ones, even if there is no explicit binding: we distinguish between the reception of a known name, i.e. a name already in $\mathcal{N}g$, and of a fresh one; in the latter case, the index is augmented. This allows us to give a finite representation of an infinite number of transitions. Notice that we do not need an

exponential to model input, as in Definition 2.3.5, because NCPi input already has an early semantics.

Proposition 4.3.3. *B_n is accessible and preserves weak pullbacks.*

This immediately follows from Proposition 4.2.5 and the fact that products, coproducts and composition preserve such properties. The consequence is that the category B_n-**Coalg** is well-behaved: it has a final object, and the final morphism gives two states the same semantics only if they are B_n-bisimilar.

As B_e-coalgebras are equivalent to early indexed labelled transition systems (see §2.3.4), our coalgebras are equivalent to the following kind of $\mathbf{G}_\mathrm{I}$-indexed labelled transition systems.

Definition 4.3.4 (Network conscious $\mathbf{G}_\mathrm{I}$-ILTS)**.** Given the following presheaf of labels

$$Lab_{\mathrm{nc}} := \overbrace{\mathcal{S}\times\mathcal{L}^\star\times\mathcal{S}}^{\text{service path}} + \overbrace{\mathcal{L}^\star}^{\text{complete path}} + \overbrace{\mathcal{S}\times\mathcal{N}}^{\text{known name input path}} + \overbrace{\mathcal{S}}^{\text{fresh site input path}} + \overbrace{\mathcal{S}\times\mathcal{S}\times\mathcal{S}}^{\text{fresh link input path}} + \underbrace{\mathcal{L}^\star\times\mathcal{S}\times\mathcal{N}}_{\text{free output path}} + \underbrace{\mathcal{L}^\star\times\mathcal{S}}_{\text{bound site output path}} + \underbrace{\mathcal{L}^\star\times\mathcal{S}\times\mathcal{S}\times\mathcal{S}}_{\text{bound link output path}}$$

a *network conscious* $\mathbf{G}_\mathrm{I}$-ILTS ($\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS) is a $\mathbf{G}_\mathrm{I}$-ILTS $(P, \longrightarrow)$ with labels in $\int Lab_{\mathrm{nc}}$, such that:

(i) Each $g \vdash p$ has countably many transitions, all of the form:

- $g \vdash p \xrightarrow{\alpha} \delta^\bullet g \vdash p'$, with $\alpha = \bullet; W; \overline{a}(\star)$ (bound site output) or $\alpha = a\star;\bullet$ (fresh site input);
- $g \vdash p \xrightarrow{\alpha} \delta^{\bullet\to\bullet} g \vdash p'$, with $\alpha = \bullet; W; \overline{a}(\star_{bc})$ (bound link output) or $\alpha = a\star_{bc};\bullet$ (fresh link input);
- $g \vdash p \xrightarrow{\alpha} g \vdash p'$ for all the other $\alpha \in Lab_{\mathrm{nc}}(g)$.

(ii) For each morphism $\sigma\colon g \to g'$ in $\mathbf{G}_\mathrm{I}$ and each $\varphi \in \{Id, \delta^\bullet, \delta^{\bullet\to\bullet}\}$: $g \vdash p \xrightarrow{\alpha} \varphi g \vdash p'$ if and only if $g' \vdash p[\sigma] \xrightarrow{\alpha[\sigma]} \varphi(g') \vdash p'[\varphi\sigma]$ (transition are preserved and reflected by morphisms).

Proposition 4.3.5. *B_n-coalgebras are in bijective correspondence with $\mathbf{G}_I$-IL$_{nc}$TSs.*

Proof. See §A.2.4. □

For behavioral equivalences on $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TSs, namely $\mathbf{G}_\mathrm{I}$-indexed bisimulations (Definition 2.2.19 instantiated with $\mathbf{C} = \mathbf{G}_\mathrm{I}$), we have the following correspondence.

Proposition 4.3.6. *Let (P, ρ) be a B_n-coalgebra. Then every B_n-bisimulation is equivalent to a $\mathbf{G}_I$-indexed bisimulation on the induced $\mathbf{G}_I$-IL$_{nc}$TS.*

Proof. See §A.2.5. □

As in the case of the π-calculus, the other direction holds only if the coalgebra's carrier preserves monomorphisms, which is a property that sheaves in $\mathbf{Sh}(\mathbf{G}_\mathrm{I})$ enjoy. We recall Theorem 4.2.5 of [63], adapted to our context: it is easy to see that $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TSs satisfy the relevant axioms characterizing the class of **I**-ILTSs treated in [63].

Proposition 4.3.7. *Let (P,ρ) a B_n-coalgebra. If P is a sheaf in $\mathbf{Sh}(\mathbf{G}_I)$ then every $\mathbf{G}_I$-indexed bisimulation on the induced $\mathbf{G}_I$-IL$_{nc}$TS is also a B_n-bisimulation on (P,ρ).*

Now we manufacture a sheaf out of the collections of well-formed processes. For the sake of simplicity we do not follow [28], where such a functor is obtained as the carrier of the initial algebra for a signature endofunctor, but we give an explicit definition:

$$\mathscr{N}g := \{p \text{ well-formed} \mid \mathrm{fn}(p) \subseteq \mathcal{N}g\}/\equiv \qquad \mathscr{N}(\sigma\colon g \to g') := \lambda p \in \mathscr{N}g.p\widetilde{\sigma}$$

where $\widetilde{\sigma}$ is the extension of $[\sigma]_\mathcal{N}$ to processes. For the purpose of defining the NCPi $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS, we just need the functor

$$\mathscr{N}_\mathrm{I} := \mathscr{R}\mathscr{N}$$

which only applies injective renamings. The following lemma tells us that $\mathscr{N}_\mathrm{I}$ indeed belongs to $\mathbf{Sh}(\mathbf{G}_\mathrm{I})$ (see Proposition 4.2.6).

Lemma 4.3.8. *$\mathscr{N}_I$ preserves pullbacks.*

Proof. See §A.2.6. □

The transition relation $\longrightarrow_\nu$ for our $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS is the smallest one generated by the rules in Figure 4.1, which associate indices to ordinary NCPi transitions. Actually, since there are infinitely many $g \in |\mathbf{G}_\mathrm{I}|$ such that $p \in \mathscr{N}_\mathrm{I}g$, each untyped transition has many typed counterparts. Notice the first five rules, inferring input and output paths: they collapse transitions that differ only for the fresh sent/received name to a single one, whose shape agrees with (ii) of Definition 4.3.4. This requires a "normalization step": the fresh name is replaced with $\star$ (resp. $\star_{bc}$), and the free names of p are replaced in the continuation through the colimit map involved in the definition of $\delta^\bullet$ (resp. $\delta^{\bullet\to\bullet}$). Satisfaction of condition (ii) follows from the fact that the transition system is closed under injective renamings (Proposition 3.4.1). The translation from ordinary to indexed π-calculus transition system given in [63, 3.3.3] is based on the same idea. However, it does not need the normalization step, because it uses finite sets of names as indices instead of their skeletal versions; the drawback of this is a higher number of states and transitions.

Example 4.3.9. Consider the transition

$$(x_{ab})(l_{ac}.\overline{a}c.\mathbf{0} \mid \overline{a}x_{ab}.x_{ab}.\mathbf{0}) \xrightarrow{\bullet;l_{ac};\overline{c}(x_{ab})} \overline{a}c.\mathbf{0} \mid x_{ab}.\mathbf{0}\ ;$$

The source process can be indexed by g of Example 4.2.4. Now, according to Figure 4.1, the continuation should be indexed by $\delta^{\bullet\to\bullet}g$, but we have to rename it

$$\frac{p \xrightarrow{ar;\bullet} p'}{g \vdash p \xrightarrow{ar;\bullet}_\nu g \vdash p'}\ {\scriptstyle r\in\mathcal{N}g} \qquad \frac{p \xrightarrow{ab;\bullet} p'}{g \vdash p \xrightarrow{a\star;\bullet}_\nu \delta^\bullet g \vdash p'[\star/b, old_g^\bullet]}\ {\scriptstyle b\notin\mathcal{N}g}$$

$$\frac{p \xrightarrow{al_{bc};\bullet} p'}{g \vdash p \xrightarrow{a\star_{bc};\bullet}_\nu \delta^{\bullet\to\bullet} g \vdash p'[\star_{bc}/l_{bc}, old_g^{\bullet\to\bullet}]}\ {\scriptstyle l_{bc}\notin\mathcal{N}g} \qquad \frac{p \xrightarrow{\bullet;W;\overline{a}(b)} p'}{g \vdash p \xrightarrow{\bullet;W;\overline{a}(\star)}_\nu \delta^\bullet g \vdash p'[\star/b, old_g^\bullet]}$$

$$\frac{p \xrightarrow{\bullet;W;\overline{a}(l_{bc})} p'}{g \vdash p \xrightarrow{\bullet;W;\overline{a}(\star_{bc})}_\nu \delta^{\bullet\to\bullet} g \vdash p'[\star_{bc}/l_{bc}, old_g^{\bullet\to\bullet}]} \qquad \frac{p \xrightarrow{\bullet;W;\overline{a}r} p'}{g \vdash p \xrightarrow{\bullet;W;\overline{a}r}_\nu g \vdash p'}$$

$$\frac{p \xrightarrow{a;W;b} p'}{g \vdash p \xrightarrow{a;W;b}_\nu g \vdash p'} \qquad \frac{p \xrightarrow{\bullet;W;\bullet} p'}{g \vdash p \xrightarrow{\bullet;W;\bullet}_\nu g \vdash p'}$$

Figure 4.1: Rules generating the transitions of $p \in \mathcal{N}_I g$ in the NCPi $\mathbf{G}_I$-IL$_{nc}$TS.

via $[a''/a, b''/b, c''/c, l''_{a''c''}/l_{ac}, \star_{a''b''}/x_{ab}]$. The resulting transition in the NCPi $\mathbf{G}_I$-IL$_{nc}$TS is

$$g \vdash (x_{ab})(l_{ac}.\overline{a}c.\mathbf{0} \mid \overline{a}x_{ab}.x_{ab}.\mathbf{0}) \xrightarrow{\bullet;l_{ac};\overline{c}(\star_{a''b''})} \delta^{\bullet\to\bullet} g \vdash \overline{a''}c''.\mathbf{0} \mid \star_{a''b''}.\mathbf{0}\ .$$

Finally, we have the following result, which relates B_n-bisimulations on the NCPi B_n-coalgebra and a class of network conscious bisimulations.

Theorem 4.3.10. *$\mathbf{G}_I$-indexed bisimulations on $(\mathcal{N}_I, \longrightarrow_\nu)$ are in bijection with:*

(i) B_n-bisimulations on the corresponding B-coalgebra;

(ii) network conscious bisimulations closed under injective renamings.

Proof. See §A.2.7. □

In particular, we have that the greatest $\mathbf{G}_I$-indexed bisimulation, the B_n-bisimilarity and $\sim^{NC}$ are all equivalent, thanks to Proposition 3.4.3.

4.4 History dependent automata

HD-automata are coalgebras with states in *named-sets* [18], that are sets whose elements are equipped with a symmetry group over finite sets of names. They have two main features:

- one single state can represent the whole orbit of its symmetry;

- the names of each state are *local*, related to those of other states via suitable mappings.

Both are important for applying finite state methods, such as minimization and model-checking, to nominal calculi. In particular, the latter point captures *deallocation*: map between states can discard unused names and "compact" remaining ones, much like *garbage collectors* do for memory locations.

In coalgebras on presheaves names have *global* meanings, and an allocating transition always adds a globally fresh name to its target state. This may generate infinite-state transition systems. However, if we take only sheaves, we gain notions such as minimal support and seeds. These are used in [30] to establish the equivalence between sheaf-based coalgebras and HD-automata.

In our category $\mathbf{Sh}(\mathbf{G}_\mathrm{I})$ minimal supports and seeds exist, for essentially the same reasons explained in §2.3.2. In order to give a characterization of our coalgebras in terms of named sets, we employ the results of [17]. Here authors define a *symmetry group over a category* $\mathbf{C}$ to be a collection of morphisms in $\mathbf{C}[c,c]$, for any $c \in |\mathbf{C}|$, which is a group w.r.t. composition of morphisms. Then they take families of such groups as their notion of generalized named sets. A first result establishes the equivalence between these families and *coproducts of symmetrized representables*, that are functors of the form

$$\sum_{i\in I} \mathrm{Hom}_{\mathbf{C}}(c_i, -)_{/\Phi_i}$$

where Φ_i is a symmetry group over $\mathbf{C}$ with domain c_i, and the quotient identifies morphisms that are obtained one from the other by precomposing elements of Φ_i. These functors, in turn, are shown to be isomorphic to *wide-pullback-preserving* presheaves on $\mathbf{C}$, a wide pullback being the limit of a diagram with an arbitrary number of morphisms pointing to the same object (pullbacks are a special case, with two such morphisms). The following theorem summarizes the described results.

Theorem 4.4.1. *Let* $\mathbf{C}$ *be a category that is small, has wide pullbacks, and such that all its morphisms are monic and those in* $\mathbf{C}[c,c]$ *are isomorphisms, for every* $c \in |\mathbf{C}|$*. Then every wide-pullback-preserving* $P \in |\mathbf{Set}^{\mathbf{C}}|$ *is equivalent to a coproduct of symmetrised representables.*

Our category $\mathbf{G}_\mathrm{I}$ satisfies the hypothesis of this theorem: by Proposition 4.2.2, it is small and has wide pullbacks due to the existence of pullbacks. In fact, the diagram of a wide pullback in $\mathbf{G}_\mathrm{I}$ is formed by a finite number of morphisms, because a finite graph always has a finite number of ingoing injective homomorphisms, so its limit can be computed via binary pullbacks. Moreover, $\mathbf{G}_\mathrm{I}$ has only monos, by definition, and $\mathbf{G}_\mathrm{I}[g,g]$ clearly has only isomorphisms, for each $g \in |\mathbf{G}_\mathrm{I}|$. Finally, our presheaf of processes $\mathscr{N}_\mathrm{I}$ preserves (wide) pullbacks, so there exists an equivalent coproduct of symmetrized representables.

In other words, we can construct an HD-automaton whose states are NCPi processes. Each state is equipped with its minimal network and a group of automorphisms on that network that preserve the state's behavior. Therefore we can perform minimization and

finite-state verification on finite-control NCPi processes. For instance, we could verify that a routing algorithm does not exhibit an unwanted behavior, or that it always converges, or other properties.

Notice that $\mathbf{G}$ does not satisfy Theorem 4.4.1, due to the presence of non-monic morphisms. Therefore we cannot apply the theory of [17] to the extended coalgebra we will give in the next section.

4.5 Saturated coalgebraic semantics

Right Kan extensions provide a canonical way of closing the coalgebraic semantics under non-injective morphisms, as shown in §2.3.5. The fundamental construction here is the following adjunction

$$\mathbf{Set}^{\mathbf{G}} \underset{\mathscr{E}_R}{\overset{\mathscr{R}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Set}^{\mathbf{G}_\mathrm{I}} \tag{4.1}$$

which exists because $\mathbf{G}_\mathrm{I}$ is small and **Set** has all limits (Corollary 2.2.13). According to Theorem 2.2.12, $\mathscr{E}_R \colon \mathbf{Set}^{\mathbf{G}_\mathrm{I}} \to \mathbf{Set}^{\mathbf{G}}$ can be computed pointwise as a limit in **Set**, which can be given the following explicit description.

Proposition 4.5.1. *Given $P \colon \mathbf{G}_I \to \mathbf{Set}$, its right Kan extension along $\mathbf{G}_I \hookrightarrow \mathbf{G}$ is*

$$\mathscr{E}_R P(g) = \left\{ t \in \prod_{\sigma \colon g \to g' \in \|\mathbf{G}\|} P(g') \;\middle|\; \begin{array}{ll} \forall \sigma & : g \to g', \\ \rho & : g' \rightarrowtail g'' \in \|\mathbf{G}\| : \\ & t_\sigma[\rho]_P = t_{\rho \circ \sigma} \end{array} \right\}$$

$$\mathscr{E}_R P(\sigma) = \lambda t \in \mathscr{R}P(g).\{(t_{\sigma \circ \sigma'})_{\sigma'}\}_{\sigma' \colon g' \to g''} \qquad (\sigma \colon g \to g')$$

Proof. See §A.2.12. □

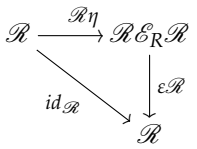
In words, $\mathscr{E}_R P(g)$ is a set of tuples with one component for each morphism from g in $\mathbf{G}$. The tuples' components are taken from P according to the corresponding morphism's codomain, and must satisfy a "closure under monos" condition, namely: selecting the σ-component of a tuple and applying $P\rho$ to it, where ρ is any applicable morphism (i.e. σ and ρ must be composable), must yield the same result as selecting the $(\rho \circ \sigma)$-component of the same tuple. As for $\mathscr{E}_R P(\sigma)$, it takes a tuple and builds another tuple out of it, whose σ'-component is the $(\sigma \circ \sigma')$-component of the original tuple.

Unit and counit of (4.1) are natural transformations $\eta \colon Id_{\mathbf{Set}^{\mathbf{G}}} \to \mathscr{E}_R\mathscr{R}$ and $\varepsilon \colon \mathscr{R}\mathscr{E}_R \to Id_{\mathbf{Set}^{\mathbf{G}_\mathrm{I}}}$. For $P \colon \mathbf{G} \to \mathbf{Set}$, $Q \colon \mathbf{G}_\mathrm{I} \to \mathbf{Set}$ and $g \in |\mathbf{G}|$, they are given by

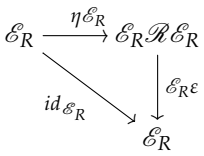
$$(\eta_P)_g = \lambda p \in Pg.\{(p[\sigma])_\sigma\}_{\sigma \colon g \to g'}$$
$$(\varepsilon_Q)_g = \lambda t \in \mathscr{R}\mathscr{E}_R Q(g).t_{id_g}$$

The intuition, in terms of processes, is that $(\eta_P)_g$ maps a process $p \in Pg$ to a tuple obtained by applying every possible renaming $\sigma\colon g \to g'$ to p. Viceversa, $(\varepsilon_Q)_g$ takes a tuple of processes in $\mathscr{R}\mathscr{E}_R Q(g)$ and extracts the one with identity index.

The well-known equations relating unit and counit ensure that the operations they perform are consistent with each other. In fact, these equations amount to the commutativity of the following diagrams

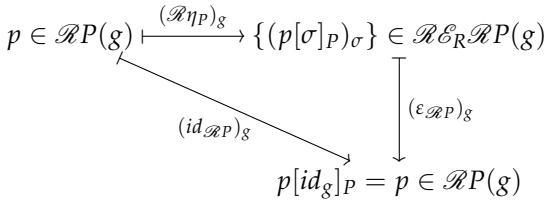


(4.2)

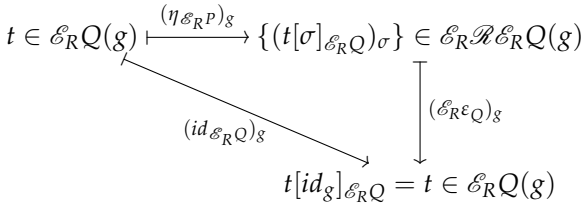


(4.3)

which, instantiated, become



(4.2)



(4.3)

In words: diagram (4.2) says that if we take a process p, rename it via η and then take the identity renaming via ε, we get p again; diagram (4.3) says the same thing about tuples.

As shown in §2.3.5, this adjunction allows us to define an extended well-behaved behavioral functor and to establish a relationship between coalgebras on this new functor and those on the old one.

Proposition 4.5.2. *The functor $\widehat{B}_n := \mathscr{E}_R B_n \mathscr{R}$ is accessible and preserves weak pullbacks.*

Proof. See §A.2.8. □

Proposition 4.5.3. *For every $P\colon \mathbf{G} \to \mathbf{Set}$, there is a bijection between $\widehat{B}_n$-coalgebras having P as carrier and B_n-coalgebras having $\mathscr{R}P$ as carrier.*

Proof. See §A.2.9. □

This bijection works as follows. Given a B_n-coalgebra $(\mathscr{R}P, \rho)$, the structure map of the corresponding $\widehat{B}_n$-coalgebra, when applied to $p \in Pg$, builds a tuple whose σ-component is the set of transitions of $p[\sigma]$ according to ρ. Viceversa, given a $\widehat{B}_n$-coalgebra (P, ϕ), one can recover a B_n-coalgebra whose structure map gives, for each $p \in \mathscr{R}P(g)$, only the id_g-component of the tuple $\phi(p)$.

$\widehat{B}_n$-coalgebras can be characterized as indexed transition systems with richer labels than those of $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TSs. This is similar to what was done in [8], whereas there is no analogous result in [28]. We call such transition systems *saturated*; this term is borrowed from [10].

Definition 4.5.4 (Saturated $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS)**.** Given the following presheaf of labels

$$Lab_{SAT} := \sum_{g' \in \mathbf{G}} \mathrm{Hom}_{\mathbf{G}}(-, g') \times Lab_{\mathrm{nc}}(g')$$

a *saturated* $\mathbf{G}_I$*-IL$_{nc}$TS* ($\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS$_{SAT}$) is a $\mathbf{G}$-ILTS $(P, \longrightarrow)$ with labels in $\int Lab_{SAT}$, such that:

(i) Transitions are of the form $g \vdash p \xrightarrow{(\sigma,\alpha)} g'' \vdash p'$, where $\sigma \in \mathbf{G}[g, g']$,$\alpha \in Lab_{\mathrm{nc}}(g')$ and $g'' \vdash p'$ is a valid continuation for α according to (ii) of Definition 4.3.4;

(ii) Transitions are such that, for all $\sigma\colon g \to g'$ in $\mathbf{G}$:

  (a) for all $\sigma'\colon g' \rightarrowtail g''$ and $\varphi \in \{Id, \delta^\bullet, \delta^{\bullet\to\bullet}\}$, $g \vdash p \xrightarrow{(\sigma,\alpha)} \varphi(g') \vdash p'$ if and only if $g \vdash p \xrightarrow{(\sigma'\circ\sigma,\alpha[\sigma']_{Lab_{\mathrm{nc}}})} \varphi(g'') \vdash p'[\varphi(\sigma')]$ (closure under monos);

  (b) $g \vdash p \xrightarrow{(\sigma'\circ\sigma,\alpha)} g'' \vdash p'$ if and only if $g' \vdash p[\sigma] \xrightarrow{(\sigma',\alpha)} g'' \vdash p'$ (transitions are preserved and reflected by morphisms);

Proposition 4.5.5. *$\widehat{B}_n$-coalgebras are in bijection with $\mathbf{G}_I$-IL$_{nc}$TS$_{SAT}$s.*

Proof. See §A.2.10. □

Behavioral equivalences for $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS$_{SAT}$s, according to Definition 2.2.19, now are indexed over $\mathbf{G}$: condition (ii), for every such a relation R, requires that, whenever $(p, q) \in R_g$, then $(p[\sigma], q[\sigma]) \in R_{g'}$, for all $\sigma\colon g \to g'$ in $\mathbf{G}$. In other words, we have closure under *all* renamings.

Now, consider the NCPi $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS $(\mathscr{N}_\mathrm{I}, \longrightarrow_\nu)$. Proposition 4.5.3 can be equivalently restated on indexed transitions systems: it allows us to derive a $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS$_{SAT}$ $(\mathscr{N}, \longrightarrow_{\nu_{SAT}})$ as follows

$$\frac{\sigma\colon g \to g' \in \|\mathbf{G}\| \qquad p \in \mathscr{N}_\mathrm{I} g \qquad g' \vdash p[\sigma] \xrightarrow{\alpha}_\nu g'' \vdash p'}{g \vdash p \xrightarrow{(\sigma,\alpha)}_{\nu_{SAT}} g'' \vdash p'}$$

Now it is clear why we use the term "saturated": $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS$_{SAT}$s are the saturated (according to [9]), but equivalent, version of the corresponding $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TSs, with contexts being the morphisms of $\mathbf{G}$. To the best of our knowledge, the fact that right-Kan-extending amounts to saturating has been first observed in [8].

Example 4.5.6. Consider the process

$$p \stackrel{\text{def}}{=} \overline{a}c.\mathbf{0} \mid l_{ab}.\mathbf{0} \mid b'(x).\mathbf{0} \ ,$$

which can be indexed by the graph

$$g \quad := \quad \begin{array}{ccc} \bullet_a & & \bullet_c \\ {\scriptstyle l}\downarrow & & \\ \bullet_b & & \bullet_b' \end{array}$$

As shown in Remark 3.4.2, its transitions are not reflected by non-injective morphisms, so they cannot be represented as a natural transformation in $\mathbf{Set}^{\mathbf{G}}$. The reason is that, if b and b' are merged, an additional transition is triggered. However, this problem disappears in $(\mathscr{N}, \longrightarrow_{\nu_{SAT}})$. To show why, we check the naturality of the corresponding $\widehat{B}_n$-coalgebra:

$$\begin{array}{ccc} p & \xrightarrow{(\nu_{SAT})_g} & \left\{ \begin{array}{l} \{\ (\bullet; l_{ab}; \overline{b}c, b'(x).\mathbf{0})\ ,\ (b'c; \bullet, \overline{a}c.\mathbf{0} \mid l_{ab}.\mathbf{0})\ , \ldots \}_{id_g}\ , \\ \left\{ \begin{array}{l} (\bullet; l_{ab}; \overline{b}c, b(x).\mathbf{0})\ ,\ (bc; \bullet, \overline{a}c.\mathbf{0} \mid l_{ab}.\mathbf{0})\ , \\ \underline{(\bullet; l_{ab}; \bullet, \mathbf{0})}\ , \ldots \end{array} \right\}_{b' \mapsto b} \end{array} , \ldots \right\} \\ {\scriptstyle [b' \mapsto b]_{\mathscr{N}}}\downarrow & & \downarrow{\scriptstyle [b' \mapsto b]_{\widehat{B}_n \mathscr{N}}} \\ p[b' \mapsto b]_{\mathscr{N}} & \xrightarrow[(\nu_{SAT})_{g'}]{} & \left\{ \left\{ \begin{array}{l} (\bullet; l_{ab}; \overline{b}c, b(x).\mathbf{0})\ ,\ (bc; \bullet, \overline{a}c.\mathbf{0} \mid l_{ab}.\mathbf{0})\ , \\ \underline{(\bullet; l_{ab}; \bullet, \mathbf{0})}\ , \ldots \end{array} \right\}_{id_{g'}} , \ldots \right\} \end{array}$$

where g' is g without the vertex b'. We only depicted some relevant transitions.

On the top right corner, with index id_g, there are an input and an output path for p as given by the original NCPi $\mathbf{G}_{\mathrm{I}}$-IL$_{\mathrm{nc}}$TS. These paths cannot be concatenated because their interaction sites are different. Then, with index $b' \mapsto b$, we have the renamed versions of the same paths, plus the underlined transition enabled by the identification of their interaction sites.

On the bottom right corner: the same transitions as above for $b' \mapsto b$, except that now they are indexed by $id_{g'}$. In fact, according to Proposition 4.5.1, the function $[b' \mapsto b]_{\widehat{B}_n \mathscr{N}}$ takes the $(b' \mapsto b \circ \sigma)$-component of the top tuple and turns it into the σ-component of the bottom tuple; in the specific case, $\sigma = id_{g'}$.

In conclusion, the fusion $b \mapsto b'$ here is harmless, because the additional transition it triggers, the underlined one, is already among those of the unrenamed process.

Finally, we want to relate the various notions of bisimulation we have introduced in this section, but the usual issue arises: the correspondence between $\widehat{B}_n$-bisimulations and $\mathbf{G}$-indexed bisimulations only holds for $\widehat{B}_n$-coalgebras and $\mathbf{G}_{\mathrm{I}}$-IL$_{\mathrm{nc}}$TS$_{SAT}$s over sheaves. However, since $\mathscr{N}_{\mathrm{I}}$ is a sheaf, Proposition 4.2.7 tells us that $\mathscr{N}$ is a sheaf as well, so we have the following result.

Theorem 4.5.7. **G**-*indexed bisimulations on* $(\mathcal{N}, \longrightarrow_{\nu_{SAT}})$ *are in bijection with:*

(i) $\widehat{B}_n$*-bisimulations on the corresponding* $\widehat{B}_n$*-coalgebra;*

(ii) network conscious bisimulations closed under all renamings.

Proof. See §A.2.11. □

By Proposition 3.5.4, the $\widehat{B}_n$-bisimilarity is also closed under input prefix. Unfortunately we do not get a full congruence, as explained in §3.5. However, if Conjecture 3.5.6 turns out to be true, we would have the following result.

Conjecture 4.5.8. $\widehat{B}_n$*-bisimilarity is a congruence for NCPi with guarded sums.*

Chapter 5

Concurrent NCPi

In this chapter we present κNCPi, the concurrent version of NCPi. We briefly introduce and motivate the main differences with NCPi.

First of all, the input primitive is more flexible: it can express the reception of a link together with its endpoints. This feature has not been included in NCPi for simplicity, but can be simulated as the sequential reception of the link's endpoints and then of the link itself. In κNCPi these actions can be performed atomically and in parallel.

Secondly, the output primitive is closer to actual routing protocols: it also specifies the *destination site*. This information is essential for routing algorithms, because they typically construct the "best" path towards a given destination address.

The last, but the most important, difference is that the semantics allows observing simultaneous actions taking place in the network, in the form of *multisets* of paths. This follows the intuition that processes should act in a truly distributed manner, without a central coordinator that imposes an interleaving order to their actions. The theoretical consequence is that bisimilarity becomes a congruence, thanks to the richer observations. This is a surprising result, given the fact that the interleaving NCPi bisimilarity is not even closed under parallel composition.

The κNCPi syntax and semantics are presented in §5.1 and §5.2, respectively. The latter section contains the main result of this chapter, namely the congruence property of κNCPi bisimilarity. In §5.3 we introduce a mechanism for controlling the inference of routing paths according to a given strategy. This allows for the implementation of routing algorithms. §5.4 applies all this machinery to a real-life routing algorithm, namely *Border Gateway Protocol* [67].

5.1 Syntax

The syntax of κNCPi processes is given in Figure 5.1. For convenience, we distinguish names that can be output or restricted (syntactic category r) and those that can be input or can be

$$
\begin{aligned}
p &::= \mathbf{0} \mid \pi.p \mid p + p \mid p \,|\, p \mid (r)p \mid A(r_1, r_2, \ldots, r_n) \\
\pi &::= \overline{a}br \mid a(s) \mid l_{ab} \mid \tau \\
r &::= a \mid l_{ab} \\
s &::= a \mid l_{(ab)} \\
A(s_1, s_2, \ldots, s_n) &\stackrel{\text{def}}{=} p \qquad i \neq j \implies \mathrm{n}(s_i) \cap \mathrm{n}(s_j) = \varnothing
\end{aligned}
$$

Figure 5.1: Syntax of κNCPi processes

α-equivalence:

$$
(a)p \equiv (a')p[a'/a] \qquad b(a).p \equiv b(a').p[a'/a] \qquad a' \# (a)p
$$

$$
\begin{aligned}
(l_{ab})p &\equiv (l'_{ab})p[l'_{ab}/l_{ab}] & l'_{ab} \# (l_{ab})p \\
a(l_{(bc)}).p &\equiv a(l'_{(b'c')}).p[l'_{b'c'}/l_{(bc)}] & b', c', l'_{b'c'} \# a(l_{(bc)}).p
\end{aligned}
$$

Unfolding law:

$$
A(r_1, \ldots, r_n) \equiv p[r_1/s_1, \ldots, r_n/s_n] \quad \text{if } A(s_1, \ldots, s_n) \stackrel{\text{def}}{=} p
$$

Figure 5.2: Structural congruence for well-formed κNCPi processes.

formal parameters of process definitions (syntactic category s); $l_{(ab)}$, belonging to the latter category, denotes a link whose endpoints are both bound and we let $\mathrm{n}(l_{(ab)}) := \{l_{ab}, a, b\}$.

Input and output prefixes have the following forms: $\overline{a}br$ means that $\overline{a}br.p$ can emit the datum r, having destination b, at a and continue as p; $a(s)$ means that $a(s).p$ can receive at a a datum to be bound to s and continue as p. The intuitive meaning of $c(l_{(ab)}).p$ is an atomic, polyadic version of $c(a).c(b).c(l_{ab}).p$. Notice that formal parameters of process definitions are of type s so, when they are links, their form is $l_{(bc)}$; actual parameters instead are of type r. The intuition, as in the input prefix, is that a formal link parameter can be instantiated with any link, not necessarily between the same endpoints.

The definition of $\mathrm{fn}(p)$ for the new constructs is

$$
\begin{aligned}
\mathrm{fn}(\overline{a}br.p) &:= \{a, b\} \cup \mathrm{n}(r) \cup \mathrm{fn}(p) \\
\mathrm{fn}(a(l_{(bc)}).p) &:= \{a\} \cup \mathrm{fn}(p) \setminus (\{b, c\} \cup \mathcal{L}_b \cup \mathcal{L}_c)
\end{aligned}
$$

Now we introduce the notion of *well-formedness* for κNCPi processes. The only additional

Paths

$$\begin{aligned}\alpha &::= a;W;b \mid \bullet;W;\bullet \mid \bullet;W;\overline{a}br \mid abr;W;\bullet \\ &\mid ab(s);W;\bullet \qquad \mathrm{n}(s)\cap(\mathrm{n}(W)\cup\{a,b\})=\varnothing \\ W &::= l_{ab} \mid W;W \mid \epsilon \\ r &::= a \mid l_{ab} \qquad s ::= a \mid l_{(ab)}\end{aligned}$$

Concurrent paths

$$\Lambda ::= \mathbf{1} \mid \alpha \mid \Lambda_1|\Lambda_2 \mid (r)\Lambda$$

Figure 5.3: Syntax of concurrent paths.

condition w.r.t. Definition 3.2.2 concerns the input prefix.

Definition 5.1.1 (Well-formed κNCPi process)**.** A κNCPi process p is *well-formed* if every subterm q satisfies requirements (i) and (ii) of Definition 3.2.2 and, moreover, (iii) $q = c(l_{(ab)}).p'$ implies $\mathrm{fn}(q) = \{c\} \cup \mathrm{fn}(p') \setminus \{a, b, l_{ab}\}$.

We introduce the following notation for substitutions: $[l'_{a'b'}/l_{(ab)}]$ stands for $[a'/a, b'/b, l'_{a'b'}/l_{ab}]$. Structural congruence, shown in Figure 5.2, is minimal: we only have α-conversion and unfolding; other axioms are moved to observations or implemented through the rules. In particular, we replace monoidicity of $|$ with analogous axioms on observations: this is quite natural, as our observations are multisets, i.e. lists quotiented by monoidal axioms. As for scope extension, it is implemented through explicit close rule. Removing axioms also allows for a simpler proof of the congruence property of bisimilarity.

5.2 Concurrent semantics

Observations for the concurrent semantics, defined in Figure 5.3, are multisets of paths, called *concurrent paths*. For the purpose of describing a more realistic network behavior, we equip paths α with some additional information:

- both input and output paths exhibit a list of links; in the case of input paths, they are the links that can be potentially traversed in order to reach the destination;
- there is a *bound input path* $ab(s);W;\bullet$, which represents the reception of a bound name; this is needed because the concurrent semantics has an explicit scope closure rule;
- paths always specify a destination site, namely b in $\bullet;W;\overline{a}br$, $abr;W;\bullet$ and $ab(s);W;\bullet$.

Moreover, we remove extrusion paths: extrusions will be represented via concurrent paths, as we will allow many paths to extrude the same name simultaneously. Concurrent paths can be of the following forms:

- the *empty concurrent path* $\mathbf{1}$ indicates that no activity is performed;
- the *singleton concurrent path* α is a concurrent path made of a single path;
- the *union* $\Lambda_1 \mid \Lambda_2$ means that the paths in Λ_1 and Λ_2 are being traversed *at the same time*;
- the *extrusion restriction* $(r)\Lambda$ indicates that r is being extruded through one or more paths in Λ.

We shall use W_α to denote the sequence of links of α and $|W_\alpha|$ to denote the set of links appearing in W_α. The functions $\mathrm{fn}(\alpha)$, $\mathrm{bn}(\alpha)$ and $\mathrm{obj}(\alpha)$ are defined on the old kinds of paths as in Table 3.1, but $\mathrm{fn}(\alpha)$ and $\mathrm{obj}(\alpha)$ now include the destination site: this is analogous to actual routing, where a payload and its destination address travel together within a packet. If α is a free input path $abr; W; \bullet$, then we have

$$\mathrm{fn}(\alpha) := \mathrm{n}(\alpha) \qquad \mathrm{bn}(\alpha) := \varnothing \qquad \mathrm{obj}(\alpha) := \mathrm{n}(r) \cup \{b\}$$

and we introduce the notation

$$\mathrm{obj}_{\mathrm{in}}(\alpha) := \mathrm{n}(r) \qquad \mathrm{obj}_{\mathrm{in}}(\alpha') := \varnothing \text{ (if } a' \text{ is not a free input path)}$$

If, instead, α is a bound input path of the form $ab(s); W; \bullet$ then

$$\mathrm{fn}(\alpha) := \mathrm{n}(\alpha) \setminus \mathrm{n}(s) \qquad \mathrm{bn}(\alpha) := \mathrm{n}(s) \qquad \mathrm{obj}(\alpha) := \{b\}.$$

Given a concurrent path Λ, the functions $\mathrm{Is}(\Lambda)$, $\mathrm{Fn}(\Lambda)$, $\mathrm{Bn}(\Lambda)$, $\mathrm{Obj}(\Lambda)$ and $\mathrm{Obj}_{\mathrm{in}}(\Lambda)$ are the extensions to multisets of the corresponding functions on single paths. They are defined as expected, but we have to be careful with the following cases:

$$\mathrm{Fn}((a)\Lambda) = \mathrm{Fn}(\Lambda) \setminus (\{a\} \cup \mathcal{L}_a) \qquad \mathrm{Bn}((a)\Lambda) = \mathrm{Bn}(\Lambda) \cup \{a\} \cup (\mathcal{L}_a \cap \mathrm{n}(\Lambda))$$

that are treated analogously for the other functions.

Observations here have a more complicated binding structure than those of NCPi, so we introduce a notion of well-formedness for them. Moreover, in order to be closer to actual routing protocol, we only admit *simple* paths, i.e. those that do not go through the same link twice.

Definition 5.2.1 (Well-formed, canonical, simple concurrent paths)**.** Let Λ be a concurrent path. Then it is:

- *well-formed* if for every subterm Λ' of the form $(a)\Lambda''$ we have $\mathrm{Fn}(\Lambda') = \mathrm{Fn}(\Lambda'') \setminus \{a\}$;

- Monoidal axioms for ";", with ϵ as identity, and for $\mid$ (plus commutativity), with $\mathbf{1}$ as identity;
- Scope extension axioms:

$$(r)(r')\Lambda \equiv_\Lambda (r')(r)\Lambda \qquad r \notin \mathrm{n}(r')$$
$$\Lambda_1 \mid (r)\Lambda_2 \equiv_\Lambda (r)(\Lambda_1 \mid \Lambda_2) \qquad r \# \Lambda_1$$

Figure 5.4: Structural congruence $\equiv_\Lambda$ for well-formed concurrent paths.

- in *canonical form* if it has the form $(R)\Theta$, where R is a sequence of restrictions and Θ does not contain extrusion restrictions (binders of the form $ab(s)$ are still allowed in Θ);
- *simple* if, for all $\alpha \in \Lambda$, each $l_{ab} \in |W_\alpha|$ appears in W_α once.

An example of non-well-formed, non-simple concurrent path is

$$(d)(\bullet; l_{ab}; l'_{ba}; l_{ab}; \overline{b}cd \mid a; l''_{ad}; d) \ ,$$

because (d) implicitly binds l''_{ad} and there are two occurrences of l_{ab} in the first path.

Simplicity is just one of the possible conditions. In general, one might want to express more complex requirements and apply static analysis methods to check them. This can be achieved through suitable type systems. For instance, QoS requirements could be expressed by associating quantitative information to links, e.g. we could say that each link l_{ab} has an average latency λ, and that l_{ab} can be added to an output path only if the path's total latency plus λ does not exceed a certain threshold.

Well-formed paths are subject to some structural congruence axioms, shown in Figure 5.4. They establish that paths are strings and concurrent paths are multisets, and that extrusion restrictions can be swapped and grouped at the outermost level. Scope extension requires some side conditions in order to avoid captures and enforce well-formedness. Clearly, any concurrent path satisfying these conditions can be converted to canonical form. We do not impose any α-conversion axiom because we cannot determine a priori which substitutions of bound names are legal for Λ: they also depend on the free names of the process doing Λ. For instance, in $(b)\overline{a}ab \mid c(x) \xrightarrow{(b)\bullet;\overline{a}ab} c(x)$ we cannot α-convert $(b)\bullet;\overline{a}ab$ to $(c)\bullet;\overline{a}ac$, because this is not a valid path for the source process. However, α-conversion of processes also affect their concurrent paths, so we have a kind of induced α-conversion.

We write $\Lambda /\!\!/ r$ for the operation that applies $/r$ to each $\alpha \in \Lambda$ if $r \notin \mathrm{Bn}(\Lambda)$, yields Λ otherwise. We use the notation $r \#_\Lambda \Lambda$, extended to sets of names as expected, to mean $r \notin \mathrm{n}(\Lambda)$.

Definition 5.2.2 (κNCPi transition system)**.** The κNCPi transition system is the smallest transition system generated by the rules in Figure 5.5, where observations are up to $\equiv_\Lambda$ and transitions are closed under $\equiv$, i.e. if $p \stackrel{\Lambda}{\Rightarrow} q$, $p \equiv p'$, $q \equiv q'$ and $\Lambda \equiv_\Lambda \Lambda'$, then $p' \stackrel{\Lambda'}{\Rightarrow} q'$.

Axioms (OUT) and (IN) generate output and input paths of length zero. Such input paths represent the reception of a datum at its destination, so their reception and destination site coincide. We also have (BIN), which gives input paths with bound placeholder. Axioms (INT) and (LINK) are the same as the interleaving case.

The axiom (IDLE) infers a "no-op" transition, enabling the parallel composition of processes to behave in an interleaving style.

The rule (SUM-L) is obvious. It has a right counterpart, because we do not have a structural congruence axiom for commutativity of $+$. This rule is omitted in Figure 5.5.

The rule (RES) and (OPEN) are an obvious extension of those in Figure 3.5. Notice that (OPEN) allows one to "extrude" the destination site: the intuition is that we can use global resources to send or receive a datum to/from a local site, which becomes global if the communication is not complete.

The rule (PAR) makes the union of two concurrent paths, but only if bound names of each concurrent path are fresh w.r.t. the other process and do not appear in the other concurrent path. This last condition avoids inferring transitions where the extruded name is free in the receiving process's continuation even if it has not been actually received, which might cause incorrect behaviors. For instance, consider the processes

$$p \stackrel{\text{def}}{=} (b)\bar{a}ab.b(c).p' \qquad q \stackrel{\text{def}}{=} a(d).\bar{d}de.q'$$

and suppose the following transition is allowed

$$p \mid q \xrightarrow{(b)\bullet;\bar{a}ab \mid aab;\bullet} b(c).p' \mid \bar{b}be.q'\{b/d\} \ .$$

Now the two components of the continuation can synchronize on b even if its scope extension has not actually been accomplished, which is clearly incorrect.

The remaining rules are used to synchronize processes. The synchronization is performed in two steps:

(i) paths of parallel processes are collected through the rule (PAR);

(ii) (COM), (SRV-IN), (SRV-OUT) and (SRV-SRV) pick two compatible paths from the resulting multiset and replace them with their concatenation, without modifying the source process; in other words, these rules synchronize two subprocesses of the source process. All other paths remain in the resulting multiset, including other occurrences of those that have been concatenated.

The rule (COM) covers all kinds of communications, yielding a complete path. In the case of extrusions, input placeholders in the continuation are replaced with extruded names. These

(IN) $a(s).p \xrightarrow{aar;\bullet} p[r/s]$ (BIN) $a(s).p \xrightarrow{aa(s);\bullet} p$ (OUT) $\overline{a}br.p \xrightarrow{\bullet;\overline{a}br} p$ (IDLE) $p \overset{\mathbf{1}}{\Rightarrow} p$

(INT) $\tau.p \overset{\bullet;\bullet}{\Rightarrow} p$ (LINK) $l_{ab}.p \xrightarrow{a;l_{ab};b} p$

(SUM-L) $$\frac{p \overset{\Lambda}{\Rightarrow} p'}{p+q \overset{\Lambda}{\Rightarrow} p'}$$

(RES) $$\frac{p \overset{\Lambda}{\Rightarrow} q}{(r)p \xrightarrow{\Lambda /\!\!/ r} (r)q}$$ $r \notin \mathrm{Obj}(\Lambda) \cup \mathrm{Bn}(\Lambda) \cup \mathrm{Is}(\Lambda)$

(OPEN) $$\frac{p \overset{\Lambda}{\Rightarrow} q}{(r)p \xrightarrow{(r)(\Lambda /\!\!/ r)} q}$$ $r \in \mathrm{Obj}(\Lambda) \setminus (\mathrm{Is}(\Lambda) \cup \mathrm{Obj}_{\mathrm{in}}(\Lambda))$

(PAR) $$\frac{p_1 \overset{\Lambda_1}{\Rightarrow} q_1 \quad p_2 \overset{\Lambda_2}{\Rightarrow} q_2}{p_1 \mid p_2 \xrightarrow{\Lambda_1 \mid \Lambda_2} q_1 \mid q_2}$$ $\mathrm{Bn}(\Lambda_i) \# p_{3-i}$; $\mathrm{Bn}(\Lambda_i) \#_\Lambda \Lambda_{3-i}$; i=1,2

(COM) $$\frac{p \xrightarrow{(R)\ (\bullet;W;\overline{a}br \mid ab'x;W';\bullet \mid \Theta)} q}{p \xrightarrow{(R')\ (\bullet;W;W';\bullet \mid \Theta)} (R'')\ q(\sigma_b \circ \sigma_r)}$$ $R'=R \cap \mathrm{Obj}(\Theta)$; $R''=(R \setminus R') \cap (\{b\} \cup \mathrm{n}(r))$; see tables (b) and (c)

(SRV-IN) $$\frac{p \xrightarrow{(R)\ (a;W;b \mid bcx;W';\bullet \mid \Theta)} q}{p \xrightarrow{(R)\ (acx;W;W';\bullet \mid \Theta)} q}$$

(SRV-OUT) $$\frac{p \xrightarrow{(R)\ (\bullet;W;\overline{a}br \mid a;W';c \mid \Theta)} q}{p \xrightarrow{(R)\ (\bullet;W;W';\overline{c}br \mid \Theta)} q}$$

(SRV-SRV) $$\frac{p \xrightarrow{a;W;b \mid b;W';c \mid \Lambda} q}{p \xrightarrow{a;W;W';c \mid \Lambda} q}$$

The concurrent path inferred by (COM), (SRV-IN), (SRV-OUT) and (SRV-SRV) must be simple.

(a)

• $b' = b$ • $\sigma_b = id$	• $b, b' \in R$ • $\sigma_b = [b/b']$

(b)

• $r \notin R$ • $x = r$ • $\sigma_r = id$	• $r \in R$ • $x = (s)$ • $\sigma_r = [r/s]$

(c)

Figure 5.5: κNCPi SOS rules: (a)shows the SOS rules; (b) and (c) are the possible configurations for (COM). Any pair of configurations, one from (b) and one from (c), is valid (four possibilities).

names are removed from the resulting transition's label, provided that there are no other paths extruding them.

The rules (SRV-IN), (SRV-OUT) and (SRV-SRV) allow extending a path with a service path. An alternative could be having a rewrite system on labels, but, since our proofs manipulate derivation trees, we prefer path extension to be performed by explicit inference rules.

The premises of (COM), (SRV-IN) and (SRV-OUT) must have their concurrent paths in canonical form: this is always possible, thanks to (PAR) side conditions.

Example 5.2.3 (Interleaving behavior)**.** Consider the process

$$\overline{a}ab.\mathbf{0} \mid a(c).\mathbf{0}$$

which can be regarded as the π-calculus process $\overline{a}b.\mathbf{0} \mid a(c).\mathbf{0}$. Its interleaving behavior can be inferred as follows

$$\dfrac{\dfrac{(\text{OUT})\ \overline{}}{\overline{a}ab.\mathbf{0} \xRightarrow{\bullet;\overline{a}ab} \mathbf{0}} \quad \dfrac{(\text{IDLE})\ \overline{}}{a(c).\mathbf{0} \overset{\mathbf{1}}{\Rightarrow} a(c).\mathbf{0}}}{(\text{PAR})\ \overline{a}ab.\mathbf{0} \mid a(c).\mathbf{0} \xRightarrow{\bullet;\overline{a}ab \mid \mathbf{1} \equiv_\Lambda \bullet;\overline{a}ab} a(c).\mathbf{0}}$$

This is analogous to the π-calculus transition $\overline{a}b.\mathbf{0} \mid a(c).\mathbf{0} \xrightarrow{\overline{a}b} a(c).\mathbf{0}$.

Example 5.2.4 (Multiple extrusions of the same name)**.** Consider the process

$$(a)(\overline{b}ca.p_1 \mid \overline{d}ea.p_2) \mid l_{bc}.p_3 \mid c(x).p_4$$

which extrudes a twice and suppose that a and x are different from all other free and bound names. If only one output is performed, we have the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{OUT})\ \overline{}}{\overline{b}ca.p_1 \xRightarrow{\bullet;\overline{b}ca} p_1} \quad \dfrac{(\text{IDLE})\ \overline{}}{\overline{d}ea.p_2 \overset{\mathbf{1}}{\Rightarrow} \overline{d}ea.p_2}}{(\text{PAR})\ \overline{b}ca.p_1 \mid \overline{d}ea.p_2 \xRightarrow{\bullet;\overline{b}ca \mid \mathbf{1} \equiv_\Lambda \bullet;\overline{b}ca} p_1 \mid p_2}}{(\text{OPEN})\ (a)(\overline{b}ca.p_1 \mid \overline{d}ea.p_2) \xRightarrow{(a)\bullet;\overline{b}ca} p_1 \mid p_2} \quad \dfrac{\dfrac{(\text{LINK})\ \overline{}}{l_{bc}.p_3 \xRightarrow{b;l_{bc};c} p_3} \quad \dfrac{(\text{BIN})\ \overline{}}{c(x).p_4 \xRightarrow{cc(x);\bullet} p_4}}{(\text{PAR})\ l_{bc}.p_3 \mid c(x).p_4 \xRightarrow{b;l_{bc};c \mid cc(x);\bullet} p_3 \mid p_4}}{(\text{SRV-IN})\ l_{bc}.p_3 \mid c(x).p_4 \xRightarrow{bc(x);l_{bc};\bullet} p_3 \mid p_4}}{(\text{PAR})\ (a)(\overline{b}ca.p_1 \mid \overline{d}ea.p_2) \mid l_{bc}.p_3 \mid c(x).p_4 \xRightarrow{(a)\bullet;\overline{b}ca \mid bc(x);l_{bc};\bullet \equiv_\Lambda (a)(\bullet;\overline{b}ca \mid bc(x);l_{bc};\bullet)} p_1 \mid p_2 \mid p_3 \mid p_4}}{(\text{COM})\ (a)(\overline{b}ca.p_1 \mid \overline{d}ea.p_2) \mid l_{bc}.p_3 \mid c(x).p_4 \xRightarrow{\bullet;l_{bc};\bullet} (a)((p_1 \mid p_2 \mid p_3 \mid p_4)[a/x]) = (a)(p_1 \mid p_2 \mid p_3 \mid p_4[a/x])}$$

If also the other output is executed, i.e. (IDLE) on the top is replaced by

$$(\text{OUT})\ \dfrac{\overline{}}{\overline{d}ea.p_2 \xRightarrow{\bullet;\overline{d}ea} p_2}$$

then the last step becomes

$$(\text{COM})\ \dfrac{(a)(\overline{b}ca.p_1 \mid \overline{d}ea.p_2) \mid l_{bc}.p_3 \mid c(x).p_4 \xRightarrow{(a)(\bullet;\overline{b}ca \mid \bullet;\overline{d}ea \mid bc(x);l_{bc};\bullet)} p_1 \mid p_2 \mid p_3 \mid p_4}{(a)(\overline{b}ca.p_1 \mid \overline{d}ea.p_2) \mid l_{bc}.p_3 \mid c(x).p_4 \xRightarrow{(a)(\bullet;\overline{d}ea \mid \bullet;l_{bc};\bullet)} p_1 \mid p_2 \mid p_3 \mid p_4[a/x]}$$

The name a is global in the continuation because one bound output path does not have a matching input path, so the scope of a cannot be closed. The missing input path could be provided by a process of the form

$$l'_{de}.p_5 \mid e(y).p_6 \ ,$$

in which case the last step of derivation would be

$$(\text{COM})\ \frac{\ldots \mid l'_{de}.p_5 \mid d(y).p_6 \xrightarrow{(a)(\bullet;\bar{b}ca \mid \bullet;\bar{d}ea \mid bc(x);l_{bc};\bullet \mid de(y);l'_{de};\bullet)} p_1 \mid p_2 \mid p_3 \mid p_4 \mid p_5 \mid p_6}{\ldots \mid l'_{de}.p_5 \mid d(y).p_6 \xrightarrow{\bullet;l_{bc};\bullet \mid \bullet;l'_{de};\bullet} (a)(p_1 \mid p_2 \mid p_3 \mid p_4[a/x] \mid p_5 \mid p_6[a/y])}$$

The following proposition states that the transition system generated by these rules is well-behaved. Notice that simplicity of paths is forced by the rules.

Proposition 5.2.5. *If $p \stackrel{\Lambda}{\Rightarrow} q$ then Λ is simple and well-formed, and q is well-formed.*

Proof. See §A.3.1. □

The behavioral equivalence for κNCPi processes is called *concurrent network conscious bisimilarity*, and is an obvious extension of Definition 3.3.4.

Definition 5.2.6 (Concurrent network conscious bisimulation)**.** A binary, symmetric and reflexive relation $\mathcal{R}$ is a *concurrent network conscious bisimulation* if $(p,q) \in \mathcal{R}$ and $p \stackrel{\Lambda}{\Rightarrow} p'$, with $\mathrm{Bn}(\Lambda) \# q$, implies that there is q' such that $q \stackrel{\Lambda}{\Rightarrow} q'$ and $(p',q') \in \mathcal{R}$. The bisimilarity is the largest such relation and is denoted by $\sim_\kappa^{NC}$.

Theorem 5.2.7. *$\sim_\kappa^{NC}$ is a congruence with respect to all κNCPi operators.*

Proof sketch. We have to prove that $\sim_\kappa^{NC}$ is closed under each operator. The difficult case is the input prefix, since a renaming, possibly not injective, is involved. The idea behind the proof is that, even though a renaming σ may enable some (COM), (SRV-IN), (SRV-OUT) or (SRV-SRV) rules in the proof of a transition of $p\sigma$, the paths these rules concatenate are renamed versions of paths already observable from p, and thus from every q bisimilar to p. We give an overview of the proof steps. The full proof can be found in §A.3.2.

(i) We prove that, given any transition $p \stackrel{\Lambda}{\Rightarrow} q$ and renaming σ such that $\Lambda\sigma$ is simple, we have $p\sigma \stackrel{\Lambda\sigma}{\Rightarrow} q\sigma$.

(ii) We give a result about the possibility of permuting rules in proofs with a certain shape. Consider a transition $p \stackrel{\Lambda}{\Rightarrow} q$ with proof Π, and suppose that this proof ends with the application of a rule that concatenates two paths α_1 and α_2 (namely one among (COM), (SRV-IN), (SRV-OUT) or (SRV-SRV)) , followed by the application of a non-concatenating rule (all the other ones). Moreover, suppose that the common interaction sites of α_1 and α_2 are free in p. Then the last two rules of Π can be permuted. In fact, if the latter rule is (OPEN) or (RES), the requirement about interaction sites ensures that they are

not involved in any of the rule's side conditions, so swapping the order of binding and concatenation is allowed. The result is the proof for a transition $p' \overset{\Lambda}{\Rightarrow} q^*$ such that $p' \equiv p$ and q^* is q with some unguarded restrictions at the outmost level (scope extension is not an axiom, but we show that it is included in the bisimilarity). In fact, when the operation just described is applied to an instance of (COM), the application of this rule is delayed, so the scope closures it infers may embrace more processes.

(iii) We prove that, for any σ and p, if $p\sigma \overset{\Lambda}{\Rightarrow} q$ has proof Π, and interaction sites or objects of paths concatenated throughout Π are not in the image of (the non-identity part of) σ, then we can recover a transition $p \overset{\Lambda'}{\Rightarrow} q'$ such that $\Lambda'\sigma = \Lambda$ and $q'\sigma = q$. In other words, if σ did not enable any concatenations in Π, then $p\sigma \overset{\Lambda}{\Rightarrow} q$ is the renamed version of a transition of p.

(iv) We show that, given a transition $p\sigma \overset{\Lambda}{\Rightarrow} q$ with proof Π, we can always recover a transition $p \overset{\Lambda'}{\Rightarrow} q'$ and a sequence of inference steps from $p\sigma \overset{\Lambda'\sigma}{\Longrightarrow} q'\sigma$ to $p\sigma \overset{\Lambda}{\Rightarrow} q^*$, where q^* is q with some unguarded restrictions at the outmost level. The idea is to collect all the concatenation steps enabled by σ at the bottom of the proof via (ii) (some restrictions may float, that's why we have q^*). This produces a new proof where we have an upper part ending with a transition that satisfies (iii), thus of the form $p\sigma \overset{\Lambda'\sigma}{\Longrightarrow} q'\sigma$, for some Λ' and q'; and a bottom part made of concatenations steps starting from this transition, thus satisfying our claim.

(v) Finally, we prove that we can construct a bisimulation $\mathcal{R}$ containing all the pairs $(p\sigma, q\sigma)$, with $p \sim_\kappa^{NC} q$, which is also closed under restrictions (i.e. $((r)p, (r)q) \in \mathcal{R}$ whenever $(p, q) \in \mathcal{R}$) and contains scope extension (i.e. if q is p with some restrictions brought at the top level, then $(p, q) \in \mathcal{R}$). In fact, given $(p, q) \in \mathcal{R}$ and $p\sigma \overset{\Lambda}{\Rightarrow} p'$, we can use (iv) to get a transition $p \overset{\Lambda'}{\Rightarrow} p''$ and a sequence Ω of rule instances that concatenate pairs of paths. Such transition can be simulated by $q \overset{\Lambda'}{\Rightarrow} q'$, by definition of $\mathcal{R}$. Using (i) we get $q\sigma \overset{\Lambda'\sigma}{\Longrightarrow} q'\sigma$ and, applying a suitably adapted version of Ω, we get $q\sigma \overset{\Lambda}{\Rightarrow} (R)(q'\sigma)\sigma'$, where R and σ' are added by instances of (COM) treating extrusions. Since Ω has the same effect on $p''\sigma$, i.e. it binds R and applies σ', we have that $(R)(p''\sigma)\sigma'$ is p' with some restrictions collected at the topmost level. Now, since $p'' \sim_\kappa^{NC} q'$, by definition of $\mathcal{R}$, and $\mathcal{R}$ is closed under under scope extension and restrictions, we can conclude $(p', (R)(q'\sigma)\sigma') \in \mathcal{R}$.

□

This result allows us to equip the π-calculus with a concurrent semantics. In fact, we can characterize π-calculus processes via a syntactic restriction, as done in Definition 3.3.5.

Definition 5.2.8 (Concurrent linkless NCPi)**.** We call *concurrent linkless NCPi* (κNCPi$_{\text{-}\ell}$) the subcalculus of κNCPi where:

- no links appear in processes;
- every occurrence of the output prefix is of the form $\overline{a}ab$.

Then, we have the following correspondences. A π-calculus output $\overline{a}b$ becomes $\overline{a}ac$, if used as prefix, or $\bullet;\overline{a}ab$, if used as action; and the input action ax becomes $aax;\bullet$, $x \in \{b,(b)\}$. The π-calculus rules for parallel composition are emulated by two steps of derivations: for (PAR), this is shown Example 5.2.3; for (COM) we have

$$\text{(COM)}\ \frac{p \xrightarrow{\overline{a}b} p' \qquad q \xrightarrow{ab} q'}{p \mid p' \xrightarrow{\tau} q \mid q'} \quad \longmapsto \quad \text{(COM)}\ \frac{\text{(PAR)}\ \dfrac{p \overset{\bullet;\overline{a}ab}{\Longrightarrow} p' \qquad q \overset{aab;\bullet}{\Longrightarrow} q'}{p \mid q \overset{\bullet;\overline{a}ab \mid aab;\bullet}{\Longrightarrow} p' \mid q'}}{p \mid q \overset{\bullet;\bullet}{\Longrightarrow} p' \mid q'}$$

In addition to the ordinary π-calculus behavior, in κNCPi-$_\ell$ we also have concurrent observations, which lead to the following result.

Corollary 5.2.9 (of Theorem 5.2.7)**.** *The bisimilarity on the concurrent π-calculus transition system is a congruence.*

Another evidence of this result is the classical counterexample not applying: we have $\overline{a}ar \mid a(x) \not\sim_\kappa^{NC} \overline{a}ar.a(x) + a(x).\overline{a}ar$, because

$$\overline{a}ar \mid a(x) \xrightarrow{\bullet;\overline{a}ar \mid aar;\bullet} \mathbf{0} \qquad \overline{a}ar.a(x) + a(x).\overline{a}ar \not\xrightarrow{\bullet;\overline{a}ar \mid aar;\bullet}$$

This result is analogous to that in [43] but, as already mentioned, there the synchronization mechanism is not faithful to the π-calculus: in [43] the synchronization channel is observed unless restricted, for instance $\overline{a} \mid a \overset{\tau_a}{\Rightarrow} \mathbf{0}$, while for our calculus $\overline{a} \mid a \overset{\bullet;\bullet}{\Longrightarrow} \mathbf{0}$, which corresponds to τ. Comparisons with other concurrent semantics for the π-calculus can be found in chapter 7.

5.3 Implementing routing algorithms

Actual routing algorithms usually build a specific path for each sender-destination pair, for instance the shortest path between them. However, our semantics non-deterministically generates all possible paths. In order to make this generation deterministic, we introduce *forwarding predicates*.

Definition 5.3.1 (Forwarding predicate)**.** A *forwarding predicate* is a boolean-valued function

$$\varphi\colon \mathcal{L} \times \mathcal{S} \times \mathit{Proc} \to \{\texttt{true}, \texttt{false}\}$$

where *Proc* is the collection of κNCPi processes.

Intuitively, $\varphi(l_{de}, c, p)$ tells wether p can use l_{de} to forward a datum that has destination c. In this way, for instance, we could exclude non-optimal links according to some metric

(cost, latency, distance, and others). SOS rules that derive extended input/output paths are modified accordingly.

$$\left.\begin{array}{ll} \text{(SRV-IN)} & \dfrac{p \xRightarrow{(R)\ (a;W;b \mid bcx;W';\bullet \mid \Theta)} q}{p \xRightarrow{(R)\ (acx;W;W';\bullet \mid \Theta)} q} \\[2ex] \text{(SRV-OUT)} & \dfrac{p \xRightarrow{(R)\ (\bullet;W';\overline{a}cr \mid a;W;b \mid \Theta)} q}{p \xRightarrow{(R)\ (\bullet;W';W;\overline{b}cr \mid \Theta)} q} \end{array}\right\} \quad \forall l_{de} \in |W| : \varphi(l_{de}, c, p)$$

These rules are sound, because they infer a subset of the paths that the original rules infer.

5.4 Example: a routing protocol

Here we give a non-trivial example of how κNCPi can be used to model a routing protocol, similar to *Border Gateway Protocol* (BGP) [67]. This protocol assumes that the network is composed of disjoint groups of networks, each referring to a single administrative authority, called *Autonomous Systems* (AS). Some of the ASs' routers act as *gateways* between the AS they belong to and other networks. The protocol takes care of the routing mechanism between ASs in a distributed manner: each gateway has a *routing table*, filled by the protocol, whose entries specify which is the next hop along the "best" path towards some destination; this information will be used to forward the incoming data.

In our model, both routers and hosts are represented as sites, and network connections are represented as links. The whole network is modelled as the parallel composition of some autonomous systems plus the connections among them (parameters of a recursive definition are omitted when unimportant)

$$\mathtt{Net} \stackrel{\text{def}}{=} \mathtt{AS}_1 \mid \ldots \mid \mathtt{AS}_k \mid \mathtt{Overlay} \qquad \mathtt{Overlay} \stackrel{\text{def}}{=} \ldots \mid \mathtt{L}(l^i_{g_i h_i}) \mid \ldots$$

$$\mathtt{L}(l_{(xy)}) \stackrel{\text{def}}{=} l_{xy}.\mathtt{L}(l_{xy})$$

Here $\mathtt{L}(l^i_{gh})$ is a process that recursively offers a transportation service over l^i_{gh} from gateway g to h. We denote by G the set $\text{fn}(\mathtt{Overlay}) \cap \mathcal{S}$, which contains the gateways.

An autonomous system $\mathtt{AS}_k$ is

$$\mathtt{AS}_k \stackrel{\text{def}}{=} (\mathscr{L}_k)(\mathtt{L}_k \mid \mathtt{A}_k) \qquad \mathtt{L}_k \stackrel{\text{def}}{=} \ldots \mid \mathtt{L}(l^{i,k}_{ab}) \mid \ldots$$

where $\mathscr{L}_k = \text{fn}(L_k) \cap \mathcal{L}$ are the *local links of* $\mathtt{AS}_k$, invisible to any other AS. We have two components: $\mathtt{L}_k$, which keeps providing the local services, and $\mathtt{A}_k$, which is the parallel composition of generic processes using some sites of $\mathtt{AS}_k$ to send and receive data, i.e. of the form $a(s).p$ or $\overline{a}br.p$. We call these sites *local sites of* $\mathtt{A}_k$, denoted by $Loc(\mathtt{A}_k)$: formally they are $Loc(a(s).p) = Loc(\overline{a}br.p) = \{a\}$. The set $Loc(\mathtt{AS}_k)$ of local sites of AS_k is

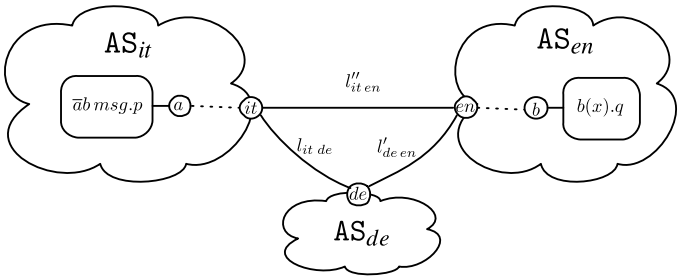


Figure 5.6: Example network.

$(\mathrm{fn}(\mathtt{L}_k) \cap \mathcal{S}) \cup Loc(\mathtt{A}_k)$. For these we require $Loc(\mathtt{AS}_i) \cap Loc(\mathtt{AS}_j) = \varnothing$, for all $i \neq j$, reflecting the fact that autonomous systems are disjoint. We write G_k for the set $Loc(\mathtt{AS}_k) \cap G$, i.e. the set of $\mathtt{AS}_k$'s gateways.

Now we want to model the routing mechanism. The routing tables are modelled as a collection of functions RT_g, one for each gateway g, such that $RT_g(x)$ is a link from g to some other gateway h, representing the next hop of the best path towards x. The forwarding is implemented via the following forwarding predicate

$$\varphi_{BGP}(l_{ab}, x, p) := \exists k : a \in G_k \implies \\ \textbf{if } x \in Loc(\mathtt{AS}_k) \textbf{ then } b \in Loc(\mathtt{AS}_k) \textbf{ else } l_{ab} = RT_a(x)$$

which means that, whenever a is a gateway of $\mathtt{AS}_k$, we have two cases: if x is in $\mathtt{AS}_k$ then a local link must be used to extend the path, otherwise the link of the overlay network specified in a's routing table. If a is not a gateway, by definition only local links are available to extend the path. These links can be used in an arbitrary way, because they are out of the scope of BGP. Instead of having a forwarding predicate, we could turn routing tables into processes, but the model would be much more complicated. In fact, one could have, for each gateway, one site for each reachable destination, and a link between two gateway sites only if they correspond to the same destination and belong to gateways involved in the optimal path toward that destination. This would rule out non-optimal complete paths.

Now, consider the network depicted in Figure 5.6. We have three ASs: an Italian one, a German one and an English one; and two processes willing to communicate from $\mathtt{AS}_{it}$ to $\mathtt{AS}_{en}$. Suppose the routing tables are such that $RT_{it}(b) = l_{it\,de}$ and $RT_{de}(b) = l'_{de\,en}$. A possible transition is

$$\mathtt{AS}_{it} \mid \mathtt{AS}_{en} \mid \mathtt{AS}_{de} \mid \mathtt{Overlay} \xrightarrow{\bullet; l_{it\,de}; l'_{de\,en}; \bullet} \mathtt{AS}'_{it} \mid \mathtt{AS}'_{en} \mid \mathtt{AS}_{de} \mid \mathtt{Overlay}\ .$$

Notice that only the part of the path between the gateways is observable.

We can give some examples of analyses that can be carried out in this scenario. We could verify that routing tables are consistent, i.e. that routing paths always reach their destination.

We could also compare paths generated by different routing algorithms, implemented via different forwarding predicates. Moreover, as mentioned, equipping links with quantitative information would allow further, more refined analyses.

Chapter 6

Case study: `Pastry`

In this chapter we model `Pastry` [59] and *Distributed Hash Tables* (DHT) using κNCPi.

We begin in §6.1 by giving an overview of `Pastry`: we describe its routing data structures and explain how routing works; then we illustrate how node joins are handled. We also formalize conditions under which `Pastry` routing converges. These will be used to prove the correctness of our model.

Then in §6.2 we introduce introduce some minor domain-specific extensions to the language. This is standard practice when modeling complex scenarios such as `Pastry`.

The following sections illustrate the model. It is organized into a *network level* and an *application level*, reflecting the fact that `Pastry` provides networking functionalities to applications.

In §6.3 we deal with the network level: we model routing data structures and their operations, reconfiguration due to joining peers, and the provision of routing functionalities to applications. Routing of messages that accomplish node joins is controlled via code in a hop-by-hop manner, because at each hop some operations must be performed. The main result of this section is that node joins preserve convergence of routing.

§6.4 is about the application level: we model a simple DHT using network-level routing to implement lookups. Unlike the previous section, routing is implemented at the SOS level, so that whole paths from the lookup invoker to the target peer can be observed. We show that routing converges also at this level: lookups always get to the correct peer.

6.1 Overview of `Pastry`

In `Pastry` peers and keys have *identifiers*, which are sequences of δ digits taken from an alphabet of σ symbols. Identifiers can be regarded as arranged in clockwise order on a *ring*, so that the greatest identifier is followed by the smallest one. An example system is shown in Figure 6.1, where identifiers are binary strings ($\sigma = 2$) of 4 digits ($\delta = 4$).

Let x, y two identifiers: we denote by $shl(x, y)$ the length of the longest prefix shared by

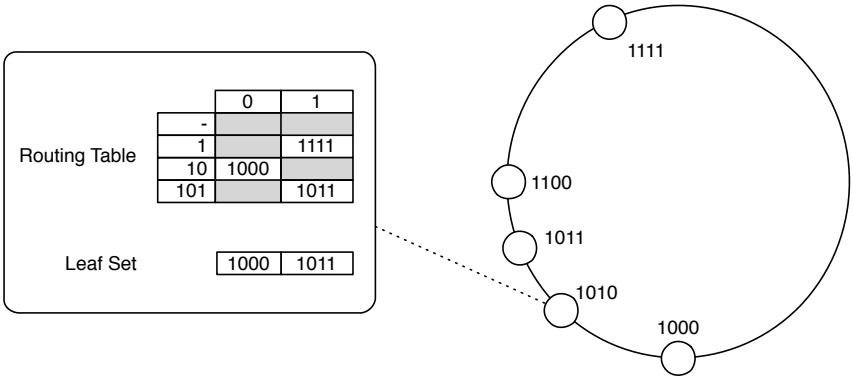


Figure 6.1: Pastry example system, with $\delta = 4$ and $\sigma = 2$. Routing data for peer 1010 are shown. Grey cells in the routing table are either empty or non non valid. For instance, the cell with coordinates $(10, 1)$ should contain an identifier starting with 101, but this is again a prefix of 1010, so the correct place for this identifier is row 101.

x and y. We define the *ring distance* d_r between x and y as the number of identifiers between x and y on the ring. Formally, if I is the size of the space of identifiers, we have

$$d_r(x,y) := \begin{cases} I - |x-y| & |x-y| > \lfloor I/2 \rfloor \\ |x-y| & \text{otherwise} \end{cases}$$

The first case happens when x and y have numerical distance greater than half the ring: then we must consider the complementary arc.

The main service provided by Pastry is *routing by key*: given a key k, Pastry delivers the message to the peer which is *responsible for* k, i.e. the one whose identifier is numerically closest to k than all other peers. Routing is implemented as follows. Each peer with identifier id maintains two data structures: a *routing table* and a *leaf-set*[1]. The routing table has $\delta \times \sigma$ entries: rows are indexed by prefixes of id, and columns by single digits; the cell $(id_1 \dots id_n, d_i)$ contains a peer whose identifier begins with $id_1 \dots id_n d_i$. The leaf-set is a set of λ peers (*leaves*) such that: $\lambda/2$ are the ones with numerically closest larger identifiers, and $\lambda/2$ are the ones with numerically closest smaller identifiers, relative to id. An example of routing table and leaf-set can be found in Figure 6.1.

Whenever a peer with identifier id receives a message with target key k, it checks whether k belongs to the leaf-set range $[\ell_m, \ell_M]$, where ℓ_m and ℓ_M are respectively the smallest and the greatest identifiers in the leaf-set. If k belongs to this interval, then id retrieves the leaf whose identifier ℓ is numerically closest to key: if $\ell = id$, then id itself is responsible for k;

[1] In [59] routing data also include a *neighborhood set*, containing references to peers that are closest to id according to a given metric. This is not relevant for routing, so it is omitted.

otherwise, the message is forwarded to ℓ. If k does not belong to the leaf-set interval, then the routing table is used: the next hop is taken from the entry in the row identified by the longest shared prefix of id and k and in the column given by the digit following such prefix in k. This ensures that the message is forwarded to a peer whose identifier has at least one more digit in common with k than id.

Example 6.1.1. Consider the peer with identifier 1010 in Figure 6.1, and suppose 1100 is responsible for the key 1101. A message from 1010 with target key 1101 is routed as follows. Since 1101 does not belong to the interval $[1000, 1011]$ spanned by the leaf-set of 1010, the routing table is used: the longest prefix shared by 1010 and 1101 is 1, so the message is forwarded to the peer in the cell $(1, 1)$, namely 1111. Once 1111 receives the message, it discovers that 1101 is in its leaf-set range, so it forwards the message to the leaf closest to 1101, that is 1100.

One important property of `Pastry` routing procedure is convergence: the message eventually gets to its destination. This is formally stated as follows.

Property 6.1.2. *The routing procedure always converges: given a message with target key k and a peer id, either id is responsible for k or can forward the message to id' such that $d_r(id', k) < d_r(id, k)$.*

Reconfiguration of the overlay happens when peers join or leave. We will only consider the case of joins, because it is the most interesting and involved ones. When a peer with identifier id_{new} intends to join the ring, it asks another peer, already in the overlay, to send a join request message on its behalf. This message is routed as its destination key were id_{new}. Each traversed peer sends its routing table back to id_{new}, so that it can initialize its own routing table; the last peer also sends its leaf-set, which contains the peers around id_{new} in the ring. Finally, id_{new} advertises its existence to all the peers in its routing table and leaf-set, which may update their own routing structures.

6.2 Notation and language extensions

We shall use some shorthands: the π-calculus output prefix $\overline{a}b$ will stand for $\overline{a}ab$; $\mathbf{new}_c(l_{ab})$ will stand for the allocation of a fresh link l_{ab} at c, namely $(l_{ab})\overline{c}l_{ab}$; we will often omit process parameters for readability.

A common pattern we will use for programming operations on data structures is the following

$$\begin{aligned}\texttt{Op} \stackrel{\text{def}}{=}\ & (x)\overline{\mathbf{op}}\ x.x(a_1).\cdots.x(a_n).\\ & \langle \textit{computation on } a_1, \ldots, a_m, \textit{ yielding results } b_1, \ldots, b_m \rangle\\ & \overline{x}\, b_1.\cdots.\overline{x}\, b_m\end{aligned}$$

This implements an operation **op** with n input parameters and m results as follows: the special site **op** provides the caller with a fresh site x, which is used to receive actual

parameters and communicate results. Freshness of x avoids interferences among concurrent calls of the same operations. Whenever the operation has one input argument and no return values, then x is not needed, so `Op` will use $\mathbf{op}(a)$ to take its argument a. For the sake of succinctness, we adopt the following notation

$$b_1, \ldots, b_m \leftarrow \mathbf{op}\langle a_1, \ldots, a_n \rangle.p$$

meaning

$$\mathbf{op}(x).\overline{x}a_1.\cdots.\overline{x}a_n.x(b_1).\cdots.x(b_m).p \ ;$$

again, if $n = 1$ and $m = 0$, the latter process becomes $\overline{\mathbf{op}}a_1.p$.

We will use *anonymous* links, i.e. the name of a link will not be specified, just its endpoints. Moreover, links will be *typed*: there will be an additional decoration that specifies the link's function. We will write $a \ \tau \ b$ for a link of type τ from a to b. Anonymity is allowed by the fact that there will be at most one link for each type between any pair of sites. We have the following types of links:

- $a \square b$ is a generic network-level link from a to b;
- $a \boxdot b$ is a link t o b in a's routing table;
- $a \boxminus b$ is a link to b in a's leaf-set;
- $a \trianglerighteq k$ is a link indicating that a is responsible for the key k in the DHT.

Endpoints between square brackets indicate that the link is argument of a link input, e.g. $c((a) \square (b))$. SOS rules are extended as expected, assuming that labels always show the type of links.

Finally, we extend the language with boolean expressions and conditional statements of the form

$$\textbf{if } bexp \textbf{ then } p \textbf{ else } q \ ;$$

we will omit the **else** branch when $q = \mathbf{0}$.

6.3 Network level

We model identifiers as site names with further structure: they are sequences of (at most) δ names taken from an alphabet $\{d_1, \ldots, d_\sigma\}$. Given an identifier a, we denote its i-th digit by a_i. For simplicity, we will use a peer name a both as its identifier and its physical address. Moreover, we assume that there is a total order relation $\prec$ on identifiers and that arithmetic operations on them are defined.

The key idea is modeling the routing table and the leaf-set of a peer as two collections of links $\mathcal{L}_{RT}$ and $\mathcal{L}_{LS}$, which form the overlay network of a peer. Our general model of a peer

with identifier a is

$$\texttt{Peer}(a, \mathcal{L}_{RT}, \mathcal{L}_{LS}) \stackrel{\text{def}}{=} (\mathcal{O}_{RT})(\mathcal{O}_{LS})\, \texttt{Control}(a, \mathcal{O}_{RT}, \mathcal{O}_{LS}) \mid \texttt{RT}(\mathcal{L}_{RT}, \mathcal{O}_{RT}) \mid \texttt{LS}(\mathcal{L}_{LS}, \mathcal{O}_{LS})$$
$$\texttt{Control}(a, \mathcal{O}_{RT}, \mathcal{O}_{LS}) \stackrel{\text{def}}{=} \texttt{JoinH}(a) + \texttt{Route}(\mathcal{O}_{RT}, \mathcal{O}_{LS})$$

The process `Control` implements the control logic of a peer: `JoinH` handles join messages and `Route` provides routing services to the application level. These operations must be performed in a mutually exclusive way, as they read and write the routing structures. This is why we have a sum between the corresponding processes. The remaining processes model the routing table (`RT`) and the leaf-set (`LS`). These data structures are equipped with some operations, which are called internally via the names in $\mathcal{O}_{RT}$ and $\mathcal{O}_{LS}$.

A `Pastry` system made of n peers with identifiers $a_1, \ldots, a_n$ is modeled as the parallel composition of peer processes. For the system Figure 6.1 we have

$$\texttt{Sys} \stackrel{\text{def}}{=} \texttt{Peer}(1000) \mid \texttt{Peer}(1010) \mid \texttt{Peer}(1011) \mid \texttt{Peer}(1100) \mid \texttt{Peer}(1111)$$

where we omitted some parameters for the sake of simplicity.

The remainder of this section is devoted to describing each component of `Peer`.

6.3.1 Data structures

The basic construct, modeling a named cell with mutable content, is the following:

$$\texttt{Cell}(c, v) \stackrel{\text{def}}{=} \bar{c}v.\texttt{Cell}(c, v) + c(v').\texttt{Cell}(c, v')$$

Output models a "get" operation, while input models a "set" operation. The cell may be empty, in which case v is either the special site **e** or a "dummy" link to this site, depending on the kind of name the cell is intended to contain.

Routing table

The routing table of a is modeled as the process

$$\texttt{RT}(a, \mathcal{L}_{\texttt{RT}}, \mathbf{next_{RT}}, \mathbf{upd_{RT}}, \mathbf{list_{RT}}) \stackrel{\text{def}}{=}$$
$$(\mathcal{C}_{\texttt{RT}})\, \texttt{Ops}_{\texttt{RT}}(\mathbf{next_{RT}}, \mathbf{upd_{RT}}, \mathbf{list_{RT}}, \mathcal{C}_{\texttt{RT}}) \;\mid\; \texttt{Cnt}_{\texttt{RT}}(\mathcal{C}_{\texttt{RT}}, \mathcal{L}_{\texttt{RT}})$$

This makes three operations available to other processes: getting the best link in the routing table towards a given key (via $\mathbf{next_{RT}}$); listing all the links in the table (via $\mathbf{list_{RT}}$); updating the table with a new peer (via $\mathbf{upd_{RT}}$). Operations are implemented in $\texttt{Ops}_{\texttt{RT}}$, while $\texttt{Cnt}_{\texttt{RT}}$ models the table's content. These processes interact via the names in $\mathcal{C}_{\texttt{RT}}$, each denoting a cell in the table.

The process $\mathtt{Cnt}_{\mathtt{RT}}$ stores the routing table's links in $\delta \times \sigma$ cells (hereafter we denote this number by ρ)

$$\begin{aligned}\mathtt{Cnt}_{\mathtt{RT}} \stackrel{\text{def}}{=}\ & \mathtt{Cell}(c_{d_1}, a \boxdot b_{1,1}) \mid \ldots \mid \mathtt{Cell}(c_{d_\sigma}, a \boxdot b_{1,\sigma}) \mid \\ & \mathtt{Cell}(c_{a_1 d_1}, a \boxdot b_{2,1}) \mid \ldots \mid \mathtt{Cell}(c_{a_1 d_\sigma}, a \boxdot b_{2,\sigma}) \mid \\ & \ldots \\ & \mathtt{Cell}(c_{a_1 \ldots a_{\delta-1} d_1}, a \boxdot b_{\delta,1}) \mid \ldots \mid \mathtt{Cell}(c_{a_1 a_2 d_\sigma}, a \boxdot b_{\delta,\sigma})\end{aligned}$$

where $c_{a_1 \ldots a_k d_i}$ denotes the cell located at $(a_1 \ldots a_k, d_i)$. For all $i = 1, \ldots, \delta - 1$, we assume that the cell in the i-th row denoted by $c_{a_1 \ldots a_{i-1} a_i}$, i.e. such that the digit identifying the column is in a, contains the dummy link $a \boxdot \mathbf{e}$, because $a_1 \ldots a_i$ concerns the $i+1$-th row.

The process $\mathtt{Ops}_{\mathtt{RT}}$ is the sum of four processes, one for each operation

$$\mathtt{Ops}_{\mathtt{RT}} \stackrel{\text{def}}{=} \mathtt{GetNext} + \mathtt{Update} + \mathtt{List}\ .$$

Computation of the next hop is performed by

$$\begin{aligned}\mathtt{GetNext} \stackrel{\text{def}}{=}\ & (x)\overline{\mathbf{next}_{\mathbf{RT}}}x.x(k).\mathbf{if}\ (k_1 \ldots k_{\delta-1} = a_1 \ldots a_{\delta-1} \wedge k_\delta \neq a_\delta)\ \mathbf{then} \\ & \qquad c_{k_1 \ldots k_\delta}((a) \boxdot (b)).\overline{x}\ a \boxdot b.\mathtt{Ops}_{\mathtt{RT}} \\ & \mathbf{else}\quad \textit{case of prefix } k_1 \ldots k_{\delta-2} \quad \ldots\end{aligned}$$

This process determines the coordinates of the cell storing the best link towards a given key k as follows. The row, i.e. the longest common prefix of a and k, is determined by comparing each prefix $a_1, \ldots, a_{i-1}$, in decreasing order of length, with the prefix of k of the same length, via a chain of nested **if-then-else**. The column is given by the first digit of k after such prefix.

The update handler replaces the content of a cell with a fresh link to a given site, provided that this site is not the empty one $\mathbf{e}$ or is already in the table. The cell's coordinates are computed as in `GetNext`:

$$\begin{aligned}\mathtt{Update} \stackrel{\text{def}}{=}\ & \mathbf{upd}_{\mathbf{RT}}(b).\mathbf{if}\ b \neq \mathbf{e}\ \mathbf{then} \\ & \qquad \mathbf{if}\ (b_1 \ldots b_{\delta-1} = a_1 \ldots a_{\delta-1} \wedge b_\delta \neq a_\delta)\ \mathbf{then} \\ & \qquad\quad c_{b_1 \ldots b_\delta}((a) \boxdot (b')).\mathbf{if}\ b \neq b'\ \mathbf{then} \\ & \qquad\qquad \mathbf{new}_a(a \boxdot b).\overline{c_{b_1 \ldots b_\delta}}\ a \boxdot b.\mathtt{Ops}_{\mathtt{RT}} \\ & \qquad \mathbf{else}\quad \textit{case of prefix } b_1 \ldots b_{\delta-2} \quad \ldots\end{aligned}$$

Process `List` is given by

$$\mathtt{List} \stackrel{\text{def}}{=} (x)\overline{\mathbf{list}_{\mathbf{LS}}}x.c_1((a) \boxdot (b_1)).\overline{x}\ a \boxdot b_1.\cdots.c_\rho((a) \boxdot (b_\rho)).\overline{x}\ a \boxdot b_\rho.\mathtt{Ops}_{\mathtt{RT}}$$

It simply lists all the links in the table.

Leaf-set

The process expression for the leaf-set of a is

$$\begin{aligned}&\mathtt{LS}(a, \mathcal{L}_{\mathtt{LS}}, \mathbf{max_{LS}}, \mathbf{min_{LS}}, \mathbf{clos_{LS}}, \mathbf{upd_{LS}}, \mathbf{list_{LS}}) \stackrel{\text{def}}{=} \\ &\quad (\mathcal{C}_{\mathtt{LS}})\ \mathtt{Ops}_{\mathtt{LS}}(a, \mathbf{max_{LS}}, \mathbf{min_{LS}}, \mathbf{clos_{LS}}, \mathbf{upd_{LS}}, \mathbf{list_{LS}}) \mid \mathtt{Cnt}_{\mathtt{LS}}(\mathcal{C}_{\mathtt{LS}}, \mathcal{L}_{\mathtt{LS}})\end{aligned}$$

This defines the following operations: getting the leaf with the lowest (via $\mathbf{min_{LS}}$) and greatest (via $\mathbf{max_{LS}}$) identifier; getting a link to the leaf closest to a given key (via $\mathbf{clos_{LS}}$); updating the leaf-set with a new leaf (via $\mathbf{upd_{LS}}$); listing all the links in the leaf-set (via $\mathbf{list_{LS}}$).

The process $\mathtt{Cnt}_{\mathtt{LS}}$ has the form

$$\begin{aligned}\mathtt{Cnt}_{\mathtt{LS}} \stackrel{\text{def}}{=}\ & \mathtt{Cell}(c_{-\lambda/2}, a \boxminus b_{-\lambda/2}) \mid \ldots \mid \mathtt{Cell}(c_{-1}, a \boxminus b_{-1}) \\ & \mid \mathtt{Cell}(c_0, a \boxdot a) \mid \mathtt{Cell}(c_1, a \boxminus b_1) \mid \ldots \mid \mathtt{Cell}(c_{\lambda/2}, a \boxminus b_{\lambda/2})\end{aligned}$$

where cells with negative (resp. positive) index contain names $\prec a$ (resp. $\succ a$). We assume that the cell with index 0 is reserved for a link from a to itself: this simplifies the implementation of some operations.

Operations are given by

$$\mathtt{Ops}_{\mathtt{LS}} \stackrel{\text{def}}{=} \mathtt{GetMin} + \mathtt{GetMax} + \mathtt{GetClosest} + \mathtt{Update} + \mathtt{List}\ .$$

The computation of the minimum is performed by the process

$$\begin{aligned}\mathtt{GetMin} \stackrel{\text{def}}{=}\ & (x)\overline{\mathbf{min_{LS}}}x.c_{-1}((a) \boxminus (b_{-1})).\cdots.c_{-\lambda/2}((a) \boxminus (b_{-\lambda/2})) \\ & \langle \mathit{find}\ m := \min\{b_{-1}, \ldots, b_{-\lambda/2}, a\}\rangle \\ & \overline{x}\, m.\mathtt{Ops}_{\mathtt{LS}}\end{aligned}$$

The comparison between leaves for finding the minimum could be implemented as a tree of nested conditional statements. The process `GetMax` is similar: it computes the maximum between $b_1, \ldots, b_{\lambda/2}, a$.

The following process returns the link pointing to the leaf closest to a given key

$$\begin{aligned}\mathtt{GetClosest} \stackrel{\text{def}}{=}\ & (x)\overline{\mathbf{clos_{LS}}}x.x(k).c_{-\lambda/2}((a) \boxminus (b_{-\lambda/2})).\cdots.c_{\lambda/2}((a) \boxminus (b_{\lambda/2})) \\ & \langle \mathit{find}\ c := \arg \min_{x \in \{b_{-\lambda/2}, \ldots, b_{\lambda/2}\} \setminus \{\mathbf{e}\}} |k - x| \rangle \\ & \overline{x}\, a \boxminus c.\mathtt{Ops}_{\mathtt{LS}}\end{aligned}$$

The addition of a new leaf is implemented by the following process

$$
\begin{aligned}
\texttt{Update} \stackrel{\text{def}}{=} \ & \mathbf{upd_{LS}}(b).c_{-\lambda/2}((a) \boxminus (b_{-\lambda/2})).\cdots.c_{\lambda/2}((a) \boxminus (b_{\lambda/2})). \\
& \quad \mathbf{if}\ b \neq b_{-\lambda/2} \wedge \cdots \wedge b \neq b_{\lambda/2} \wedge b \neq \mathbf{e}\ \mathbf{then} \\
& \qquad \mathbf{new}_a(a \boxminus b). \\
& \qquad \mathbf{if}\ b \prec a\ \mathbf{then} \\
& \qquad\quad \mathbf{if}\ b_{-\lambda/2} = \mathbf{e}\ \mathbf{then}\ \overline{c_{-\lambda/2}}\ a \boxminus b.\texttt{Ops}_{\texttt{LS}}\ \mathbf{else} \\
& \qquad\quad \cdots \\
& \qquad\quad \mathbf{if}\ b_{-1} = \mathbf{e}\ \mathbf{then}\ \overline{c_{-1}}\ a \boxminus b.\texttt{Ops}_{\texttt{LS}}\ \mathbf{else} \\
& \qquad\qquad \langle \textit{find}\ b_{\min} := \min\{b_{-\lambda/2}, \ldots, b_{-1}\}\rangle \\
& \qquad\qquad \mathbf{if}\ b_{\min} \prec b\ \mathbf{then}\ \overline{c_{\min}}\ a \boxminus b.\texttt{Ops}_{\texttt{LS}} \\
& \qquad \mathbf{else} \\
& \qquad\quad \langle \text{look for an empty cell among } c_1, \ldots, c_{\lambda/2}\rangle \\
& \qquad\quad \langle \text{if no cell is empty and } b \prec b_{\max}\text{, replace the content of } c_{\max}\rangle
\end{aligned}
$$

This process retrieves the content of all the cells and checks whether the provided site b is not already in one of them. If it is not, then creates a fresh link $a \boxminus b$ and looks for an empty cell (i.e. one containing the dummy link) among those where this link should be placed, determined according to the position of b w.r.t. a; if no empty cell exists and b is between a and the extremal leaf, then $a \boxminus b$ is placed in the cell containing the link to the extremal leaf. Let us briefly discuss the correctness of this procedure. If there is an empty cell, it means that there are not enough peers in the system. Then $a \boxminus b$ can be placed in any cell in the correct half of the leaf-set, as cells in the same half are not ordered. Otherwise $a \boxminus b$ should take part in the leaf-set, i.e. should replace one of the existing leaves, only if b falls inside the current leaf-set range. The leaf to be replaced is one of the extremal leaves, because the existence of b suggests that there are enough leaves in a smaller interval.

Finally, the process `List` is defined as its counterpart in `RT`.

6.3.2 Join Handler

The join procedure is implemented by `JoinH`. It aims at setting up routing table and leaf-set of a joining peer. This involves exchanging a variety of messages. We will represent a message of type **msg_type** and content $c_1, \ldots, c_n$ as a sequence of names **msg_type**, $c_1, \ldots, c_n$, and we will denote it as **msg_type**$[c_1, \ldots, c_n]$. Whenever we write such expression as object of an output prefix, we mean that all the elements of the message are sent in sequence.

Incoming messages are handled as follows: after receiving a name **msg_type** on the peer site a, a number of other receptions is performed, depending on the message type, to receive the message payload, and then the appropriate handler is activated. Therefore, using an

intuitive notation, we have:

$$\begin{aligned}\texttt{JoinH} \stackrel{\text{def}}{=} a(b).&\textbf{case } b : \\ &\mathbf{msg}_1[c_1, \ldots, c_{n_1}] \Rightarrow \mathit{handler}_1 \\ &\mathbf{msg}_2[d_1, \ldots, d_{n_2}] \Rightarrow \mathit{handler}_2 \\ &\ldots\end{aligned}$$

We can have the following types of messages:

- **join_req**$[b]$: join request from peer b;
- **join_ack**$[R]$: join acknowledgement, answering a join request with the list R of peers occurring in the sender's routing table.
- **join_ack_leaf**$[R, L]$: join acknowledgement with leaf-set; this is the last answer to a join request, coming from the peer whose identifier is the closest to the joining one; R and L are the peers in the sender's routing table and leaf-set.
- **join_ntf**$[b]$: join notification, informing that b has successfully joined the ring.

Handlers for these messages are shown in Figure 6.2.

A join request from a peer b is always answered with an acknowledgement via a temporary link $a \,\square\, b$. The content of such message depends on how close is a to b. If b belongs to the interval $[\ell_{\min}, \ell_{\max}]$ spanned by the leaf-set of a, then the link to the leaf d closest to b is retrieved, and we have the following cases:

- If $d \neq a$, then the join request is forwarded to d, and the join acknowledgement carries the identifiers of a's routing table. Sending the acknowledgement takes $\rho + 1$ activations of l_{ab}: one for the name denoting the message type and one for each identifier.
- if $d = a$ then the join request does not need to be further forwarded; the response acknowledgement is like in the previous case, but it also contains a's leaf-set, so that b can form its own leaf-set out of it.

If b does not belong to the leaf set, then it is forwarded according to the routing table, and the join acknowledgement again contains the routing table's entries of a.

The reception of a join acknowledgements causes the routing table and leaf-set to be updated with the content of the message. After receiving the last acknowledgement (case **join_ack_leaf**), a notifies its existence to all the peers in its routing data structures. A join notification triggers an update to both the routing table and the leaf-set of the receiving peer; additional actions may be undertaken, for instance the redistribution of hash-table entries among peers in a DHT, but these are not the concern of our model.

Finally, we give a correctness result for our implementation of the join procedure.

$$
\begin{array}{lcl}
\textbf{join\_req}[b] & \Rightarrow & (a \boxdot b) \\
 & & \ell_{\min} \leftarrow \textbf{min}_{\textbf{LS}}. \\
 & & \ell_{\max} \leftarrow \textbf{max}_{\textbf{LS}}. \\
 & & a \odot c_1, \ldots, a \odot c_\rho \leftarrow \textbf{list}_{\textbf{RT}}. \\
 & & \textbf{if } \ell_{\min} \prec b \prec \ell_{\max} \textbf{ then} \\
 & & \quad a \boxminus d \leftarrow \textbf{clos}_{\textbf{LS}}\langle b \rangle. \\
 & & \quad \textbf{if } d \neq a \textbf{ then} \\
 & & \quad\quad \overbrace{a \boxdot b \mid \ldots \mid a \boxdot b}^{\rho+1} \mid \overline{a}b\, \textbf{join\_ack}[c_1, \ldots, c_\rho] \mid \\
 & & \quad\quad a \boxminus d \mid a \boxminus d \mid \overline{a}d\, \textbf{join\_req}[b] \mid \texttt{Control} \\
 & & \quad \textbf{else} \\
 & & \quad\quad a \odot e_1, \ldots, a \odot e_\lambda \leftarrow \textbf{list}_{\textbf{LS}}. \\
 & & \quad\quad \underbrace{a \boxdot b \mid \ldots \mid a \boxdot b}_{\rho+\lambda+1} \\
 & & \quad\quad \mid \overline{a}b\, \textbf{join\_ack\_leaf}[c_1, \ldots, c_\rho, e_1, \ldots, e_\lambda] \mid \texttt{Control} \\
 & & \textbf{else} \\
 & & \quad \overbrace{a \boxdot b \mid \ldots \mid a \boxdot b}^{\rho+1} \mid \overline{a}b\, \textbf{join\_ack}[c_1, \ldots, c_\rho]. \\
 & & \quad a \odot f \leftarrow \textbf{next}_{\textbf{RT}}\langle b \rangle. \\
 & & \quad a \odot f \mid a \odot f \mid \overline{a}f\, \textbf{join\_req}[b] \mid \texttt{Control} \\
 & & \\
\textbf{join\_ack}[R] & \Rightarrow & \textbf{upd}_{\textbf{RT}}\langle R_1 \rangle. \cdots . \textbf{upd}_{\textbf{RT}}\langle R_\rho \rangle. \texttt{Control} \\
 & & \\
\textbf{join\_ack\_leaf}[R, L] & \Rightarrow & \textbf{upd}_{\textbf{RT}}\langle R_1 \rangle. \cdots . \textbf{upd}_{\textbf{RT}}\langle R_\rho \rangle. \\
 & & \textbf{upd}_{\textbf{LS}}\langle L_1 \rangle. \cdots . \textbf{upd}_{\textbf{LS}}\langle L_\lambda \rangle. \\
 & & a \odot b_1, \ldots, a \odot b_\rho \leftarrow \textbf{list}_{\textbf{RT}}. \\
 & & a \boxminus c_1, \ldots, a \boxminus c_\lambda \leftarrow \textbf{list}_{\textbf{LS}}. \\
 & & a \odot b_1 \mid a \odot b_1 \mid \overline{a}b_1\, \textbf{join\_ntf}[a] \mid \\
 & & \ldots \\
 & & \mid a \boxminus c_\lambda \mid a \boxminus c_\lambda \mid \overline{a}c_\lambda\, \textbf{join\_ntf}[a] \\
 & & \mid \texttt{Control} \\
 & & \\
\textbf{join\_ntf}[b] & \Rightarrow & \textbf{upd}_{\textbf{RT}}\langle b \rangle. \textbf{upd}_{\textbf{LS}}\langle b \rangle. \texttt{Control}
\end{array}
$$

Figure 6.2: Code handling join messages.

Theorem 6.3.1. *Consider a* `Pastry` *system with n peers $a_1, \ldots, a_n$. Let a_{n+1} be the identifier of a peer that intends to join the system. Let $\mathcal{L}_{RT}^{a_i}$ and $\mathcal{L}_{LS}^{a_i}$ be the links of a_i's routing table and leaf-set, for $i = 1, \ldots, n+1$. Then, after the join procedure for a_{n+1} has ended, the following property holds: for each key k and each a_i, either a_i is responsible for k or there is a link from a_i to b in $\mathcal{L}_{RT}^{a_i} \cup \mathcal{L}_{LS}^{a_i}$ such that b is closer to k than a_i, i.e. a_i, b and k satisfy Property 6.1.2 with $id = a_i$ and $id' = b$.*

Proof. See §A.4.1. □

6.3.3 Routing services provider

The provision of routing services to applications is performed by

$$\begin{aligned}\texttt{Route} \stackrel{\text{def}}{=} a \boxdot b_1, \ldots, a \boxdot b_\rho \leftarrow \mathbf{list_{RT}}. \sum_{b_i \neq \mathbf{e}} a \boxdot b_i.\texttt{Control} \\ + a \boxminus c_1, \ldots, a \boxminus c_\lambda \leftarrow \mathbf{list_{LS}}. \sum_{c_j \neq \mathbf{e}} a \boxminus c_j.\texttt{Control}\end{aligned}$$

It activates any link from the routing table or leaf-set, provided that it is not the dummy link (i.e. its target is not **e**).

6.4 Application level

Now we want to model routing behavior for a simple Distributed Hash Table, where observations are routing paths taken by DHT lookups.

The `Pastry` routing strategy is implemented through the forwarding predicate $\varphi_{\texttt{Pastry}}$, shown in Figure 6.3. Let us explain it. The first case allows forwarding a message with target k from a to b, via a link in a's leaf-set, provided that: there is no other leaf b' which is closer to k than b; a has two leaves b_1 and b_2, on opposite sides of (but not necessarily distinct from) a, and k is between them, i.e. k is within the leaf-set range. The second case allows a forwarding through a link in the routing table whenever there is no better link in the leaf-set and the identifier b of the reached peer shares (at least) one more digit with k than a. The third case treats links that allows reaching a key k via the peer responsible for it: it required that the the link's target is indeed k.

We can model a Distributed Hash Table over a `Pastry` system with peers $a_1, \ldots, a_n$ as follows. Suppose the DHT has m key-value pairs $\langle k_i, v_i \rangle$, and let a_{k_i} be the identifier of the peer responsible for k_i, i.e. the closest to k_i among $a_1, \ldots, a_n$. Then we have

$$\begin{aligned}\texttt{DHT} &\stackrel{\text{def}}{=} \texttt{Peer}(a_1) \mid \ldots \mid \texttt{Peer}(a_n) \mid \texttt{H} \\ \texttt{H} &\stackrel{\text{def}}{=} \texttt{Entry}(k_1, v_1, a_{k_1}) \mid \ldots \mid \texttt{Entry}(k_m, v_m, a_{k_m}) \\ \texttt{Entry}(k, v, a) &\stackrel{\text{def}}{=} a \trianglerighteq k \mid k(b).\overline{a}bv.\texttt{Entry}(k, v, a)\end{aligned}$$

$$\varphi_{\texttt{Pastry}}(l_{ab}, k, p) \;:=\; \textbf{case } l_{ab} \textbf{ of}$$

$$a \boxminus b \quad \Rightarrow \quad \left(\begin{array}{c} \forall b' \neq b : p \xRightarrow{a;a \boxminus b';b'} p' \implies d_r(k,b) < d_r(k,b') \\ \wedge \\ \exists b_1, b_2 : \left(\begin{array}{c} p \xRightarrow{a;a \boxminus b_1;b_1} p_1, p \xRightarrow{a;a \boxminus b_2;b_2} p_2 \\ \wedge \\ b_1 \preceq a \preceq b_2 \;\wedge\; b_1 \prec k \prec b_2 \end{array} \right) \end{array} \right)$$

$$a \boxdot b \quad \Rightarrow \quad \left(\begin{array}{c} \forall b' \neq b : p \xRightarrow{a;a \boxminus b';b'} p' \implies d_r(k,b) < d_r(k,b') \\ \wedge \\ shl(b,k) > shl(a,k) \end{array} \right)$$

$$a \trianglerighteq b \quad \Rightarrow \quad b = k$$

$$\left. \begin{array}{ll} \text{(SRV-IN)} & \dfrac{p \xRightarrow{(R)\ (d;W;c \mid ckx;W';\bullet \mid \Theta)} q}{p \xRightarrow{(R)\ (dkx;W;W';\bullet \mid \Theta)} q} \\ \\ \text{(SRV-OUT)} & \dfrac{p \xRightarrow{(R)\ (\bullet;W';\bar{c}kr \mid c;W;d \mid \Theta)} q}{p \xRightarrow{(R)\ (\bullet;W';W;\bar{d}kr \mid \Theta)} q} \end{array} \right\} \quad \forall l_{ab} \in |W| : \varphi_{\texttt{Pastry}}(l_{ab}, k, p)$$

Figure 6.3: `Pastry` forwarding predicate and rules.

Here H represents the DHT content as the parallel composition of processes that handle the table's entries. The idea is implementing a DHT lookup request for a key k as a message with destination k, carrying the identifier b of the sender. Upon receiving this message, the handler for $\langle k, v\rangle$ replies to b with a message containing v.

We provide an account of Property 6.1.2 in this scenario.

Lemma 6.4.1. *For every peer a and key k there is $\mathtt{DHT} \xRightarrow{a;a\rhd b;b} \mathtt{DHT}'$, where $\rhd \in \{\boxminus, \boxdot, \trianglerighteq\}$, such that either $b = k$ or b is closer to k than a, i.e. a, b, k satisfy Property 6.1.2.*

Proof. See §A.4.2. □

The following result is an immediate consequence of Lemma 6.4.1 and of the definition of $\varphi_{\mathtt{Pastry}}$. It says that, given a key k and a peer a, there always is a path from a routing a lookup request for k.

Theorem 6.4.2. *Let k be a key in the DHT and a_k the peer responsible for it. Then, for every peer a, there exists a transition*

$$\mathtt{DHT} \xRightarrow{aa_k a;a\rhd a_1;\ldots;a_n\rhd a_k;a_k\trianglerighteq k;\bullet} \mathtt{DHT}'$$

with $\rhd \in \{\boxminus, \boxdot, \trianglerighteq\}$ and $n \geq 0$.

As an example, we show how to compute a routing path in the system of Figure 6.1. For simplicity, let us consider a DHT with only one key-value pair $(1101, v)$ located at 1100:

$$\mathtt{H} \stackrel{\text{def}}{=} 1100 \trianglerighteq 1101 \mid 1101(a).\overline{1100}\, a\, v.\mathtt{H} \qquad \mathtt{DHT} \stackrel{\text{def}}{=} \mathtt{Sys} \mid \mathtt{H}$$

Consider the following process, representing a user application running at 1010

$$\mathtt{App} \stackrel{\text{def}}{=} \overline{1010}\ \ 1101\ \ 1010.1010(v').\mathtt{App}'(v') \ .$$

This sends a lookup request for the key 1101, receives the result and uses it for some computations. So we have

$$\mathtt{App} \xRightarrow{\bullet;\overline{1010}\ 1101\ 1010} 1010(v').\mathtt{App}'(v') \ .$$

The routing steps for this request are those of Example 6.1.1. In this context, they become the following ones, depicted in Figure 6.4

$$\mathtt{Peer}(1010) \xRightarrow{1010;1010\boxdot 1111;1111} \mathtt{Peer}(1010)$$
$$\mathtt{Peer}(1111) \xRightarrow{1111;1111\boxminus 1100;1100} \mathtt{Peer}(1111)$$

These paths can be concatenated with the one of App using (SRV-OUT) in Figure 6.3. The result is

$$\bullet; 1010 \boxdot 1111; 1111 \boxminus 1100; \overline{1100}\ 1101\ 1010$$

The complementary path can be inferred using (SRV-IN) in Figure 6.3

$$\mathtt{H} \xRightarrow{1100\ 1101\ 1010;1100\trianglerighteq 1101;\bullet} \overline{1100}\ 1010\ v.\mathtt{H} \ .$$

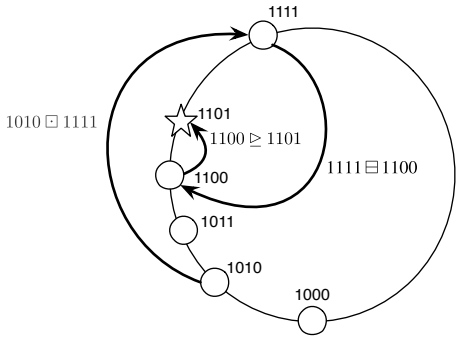


Figure 6.4: Routing path from 1010 for the key 1101 in the system of Figure 6.1.

Finally, we can concatenate all these paths through (COM) and get

$$\texttt{App} \mid \texttt{DHT} \xrightarrow{\bullet;1010\boxdot 1111;1111\boxminus 1100;1100\trianglerighteq 1101;\bullet} 1010(v').\texttt{App}(v') \mid \texttt{Sys} \mid \overline{1100}\,1010\,v.\texttt{H}$$

which exhibits the whole routing path from 1010 to 1100. Finally, assuming that the overlay network has a path back to 1010, the following configuration is reached

$$\texttt{App}(v) \mid \texttt{DHT}\ .$$

Chapter 7

Conclusions

In this thesis we presented NCPi, an extension of π-calculus with an explicit notion of network. To achieve this, we enriched the syntax with named connectors and defined a semantics whose observations are routing paths. We constructed operational models for our calculus, in terms of presheaf-based coalgebras: one characterizing the ordinary observational equivalence, with an equivalent HD-automaton, and a saturated one. Then we introduced a concurrent extension of our calculus, with additional constructs and multisets of paths as observations, and we proved that observing concurrency makes the bisimilarity of our calculus compositional. Finally, we illustrated the expressive power of our calculus by modeling network reconfiguration and routing in `Pastry` systems.

7.1 Related work

7.1.1 Other network-conscious calculi

The works most closely related to ours are [29] and [21] where network-aware extensions of $D\pi$ [35] and KLAIM[19] are presented, called respectively $D\pi_F$ and TKLAIM. KLAIM is quite far from the synchronous π-calculus, because it models a distributed tuple-space modifiable through asynchronous primitives, but an encoding to the asynchronous π-calculus exists [20].

Both $D\pi_F$ and TKLAIM are *located* process calculi, which means that processes are deployed in *locations*, modeling physical network nodes. In κNCPi, instead, processes access the network through sites, possibly more than one for each process, rather than being inside of it. However, locations can be easily introduced in κNCPi by a typing mechanism which limits the number of subject names in processes. For instance, we could introduce *location types* and associate them to sites, with the meaning that each site belongs to a location. Then, we could forbid processes where the same site occurs with different location types.

The network representations are quite different: in $D\pi_F$ locations are explicitly associated

with their connectivity via a type system, TKLAIM has a special process to represent connections, while in our calculus connections are just names, so the available network nodes and connections correspond to the standard notion of free names. This brings simpler primitives, but also a higher level of dynamics: connections can be created and passed among processes, as shown in the introductory example §3.1; this example, in our opinion, is not easily implementable in TKLAIM and $D\pi_F$.

Finally, our calculus is more programmable: processes explicitly activate transportation services over connections via the link prefix, while in the cited calculi the network is always available.

We can also cite [32, 33, 22] as examples of calculi where resources carry some extra information: they explicitly associate costs with π-calculus channels through a type system. In our case, links could also be typed in order to model services with different features, e.g. performance, costs and access rights.

7.1.2 Other presheaf models

Besides the π-calculus, other calculi have been equipped with a presheaf-based semantics: the open π-calculus in [31], where processes are indexed by structured sets of names that represent distinctions; the explicit fusion calculus in [8], where processes are indexed by fusions in the form of equivalence classes of names; and the fusion calculus in [46], where the author uses the same presheaf category as [28] and incorporates fusions in the behavioral functor.

7.1.3 Other concurrent semantics for the π-calculus

There are other notions of concurrent semantics for the π-calculus. Some of them are obtained indirectly, via an encoding to existing concurrent semantics. In [36, 24], which inspired [43], π-calculus sequential processes are encoded as hyperedges in a *synchronized hyperedge replacement* system, where nodes model channels. The semantics is generated by means of rewrite productions and SOS rules that combine them. Many rewritings, thus many actions, can be observed happening in parallel. Observations are π-calculus actions, but they are always located, i.e. they are associated to a node; instead, we hide the node where a synchronization happens, as in the π-calculus. Moreover, it is not possible to perform two simultaneous actions on the same node, while in κNCPi this is allowed. In [13] π-calculus processes are encoded to Petri nets with inhibitor arcs: places represent the syntactic structure, and tokens the possible control flows; inhibitor arcs avoid unwanted parallel computations. The resulting semantics forbids parallel extrusions of the same name, while we allow them (see e.g. Example 5.2.4). Finally, in [49], the π-calculus semantics is emulated through a (double pushout-based) graph transformation system: as in [36], processes are encoded to hyperedges and channels as nodes; there is a special unary hyperedge indicating that the name it is connected to is unrestricted, and then can be used as subject in a

communication. Observations are those of the π-calculus, but they are computed in two steps, because much more information needs to be observed in order for rewriting to happen; this information is then discarded. This approach also accounts for parallel extrusion.

There are notions of bisimulations that take into account concurrency: in [61] π-calculus processes are equipped with locations in order to observe the degree of parallelism, but the obtained observational equivalence is not a congruence; in [11, 23] *causal relations* are used to keep track of dependencies among actions, however these relations exclude parallel extrusions. These bisimulations are rather involved, and are defined on top of the interleaving π-calculus transition system. We, instead, take the ordinary bisimilarity on a concurrent operational semantics, which makes explicit the amount of parallelism available.

7.2 Future research directions

7.2.1 Variants of NCPi

Our calculus only captures point-to-point communication, but a network could be used for more complex forms of interaction, e.g. multicast. One possible development direction might be allowing different mechanisms of message exchanging, for instance broadcast, multicast, and others.

We could also equip sites and links with pieces of information such as access rights, bandwidth, costs. For instance, just like we have source and target operations on links, we could have operations $r_{\mathcal{L}} \colon \mathcal{L} \to A$ and $r_{\mathcal{S}} \colon \mathcal{S} \to A$ that assign to each link and site, respectively, an access right in a lattice A. This information could be used to control the inference of paths.

This would be useful for modeling the Infrastructure-as-a-Service layer of Cloud Computing Systems, where users can request network resources with specific features.

Finally, in NCPi there is room for asynchronous variations. For instance, we can think of a fully asynchronous version, based on the asynchronous π-calculus, where the semantics shows single forwarding steps. This could be implemented via the following SOS rules:

$$\overline{a}br \xrightarrow{\bullet;\overline{a}br} \mathbf{0} \qquad \frac{p \xrightarrow{\bullet;\overline{a}br} p' \quad q \xrightarrow{a;l_{ac};c} q'}{p \mid q \xrightarrow{\bullet;l_{ac};\bullet} \overline{c}br \mid p' \mid q'} \qquad \frac{p \xrightarrow{\bullet;\overline{a}ar} p' \quad q \xrightarrow{aar;\bullet} q'}{p \mid q \xrightarrow{\bullet;\bullet} p' \mid q'}$$

The idea, as in the asynchronous π-calculus, is to allow only outputs without continuation. These outputs can "move" along links (second rule) and trigger a local synchronization at its destination site, provided that there is a matching input (third rule).

7.2.2 A general framework

This thesis is a first step towards a general framework for constructing operational models of resource-aware calculi. It validates the approach [28] and shows how it can be applied

to a calculus with resources that are significantly more complex than π-calculus channels. Moreover, this approach is integrated with different models that take this additional complexity into account, such as saturated transition systems and HD-automata.

Our next step will be considering the ψ-calculus [7], which is an extension of the π-calculus with nominal data types for data structures and for logical assertions representing facts about data. These can be exchanged between processes using the standard π-calculus mechanisms. Collections of nominal data could be modeled as a suitable category of resources for presheaves, and we could have allocation operators on them, e.g. modeling the introduction of a new assertion.

We also plan to investigate other extensions of the π-calculus, adding other pieces of information to sites and links, e.g. *access rights* as mentioned in §7.2.1, and study the corresponding presheaf semantics. The idea is that, since the category of resources can be constructed as a category of algebras, as we did for graphs, associating more information to resources means adding sorts (objects) and operations (morphisms) to the category describing the algebraic specification. Sorts and operations should be interpreted in suitable domains, e.g. we may want access rights to have a lattice structure.

Appendix A

Proofs

We make the following assumption about bound names of processes.

Convention A.1. When considering a collection of renamings and processes, we assume that bound names of processes are distinct from their free names and from the names involved in renamings, unless otherwise specified.

A.1 Proofs for Chapter 3

A.1.1 Proof of Proposition 3.3.3

By induction on the inference of $p \xrightarrow{\alpha} q$. The only relevant case is whenever $p \xrightarrow{\alpha} q$ is inferred through (RES). We have $p \equiv (r)p'$, $\alpha \equiv_\alpha \alpha'/r$, $q \equiv (r)q'$ and the transition is inferred from $p' \xrightarrow{\alpha'} q'$. Since $\equiv$ preserves well-formedness, $(r)p'$ is well-formed as well and so is p'. Therefore, by induction hypothesis, q' is well-formed. Now we have to show that $(r)q'$ is well-formed. Suppose it is not, then there must be $l_{ab} \in \text{fn}(q')$ such that $r = a$ or $r = b$. We show that $l_{ab} \in \text{fn}(p')$, which implies that $(r)p'$ is not well-formed, against the hypothesis. By (RES) side conditions we have that α' cannot be an output path where l_{ab} is bound or an input path with object l_{ab}. These are the only kinds of paths that can create new free names. Therefore, l_{ab} was already free in p'.

A.1.2 Proof of Theorem 3.5.2

We have to show that if $p \sim^{NC} q$ then $op(p) \sim^{NC} op(q)$, for each NCPi operator op. Most of the cases are minor adjustements of standard π-calculus proofs (see e.g. [48]). We show the cases that involve new operators and new SOS rules. For each case we define a relation $\mathcal{R}$ and prove that it is a bisimulation. We make the following simplification: for each considered $\mathcal{R}$ we assume that $(p, q) \in \mathcal{R}$ is such that $\text{bn}(p) \# q$ and viceversa, and we will only consider transitions that are inferred direclty through the rules, without using structural congruence.

This is allowed by a direct consequence of Definition 3.3.1: whenever $\mathcal{R}$ is a bisimulation, also $\equiv \mathcal{R} \equiv$ is a bisimulation.

Case $op = l_{ab}.-$**:** We have to prove that

$$\mathcal{R} = \{(l_{ab}.p, l_{ab}.q) \mid p \sim^{NC} q\} \cup \sim^{NC}$$

is a bisimulation. Clearly $l_{ab}.p$ and $l_{ab}.q$ can both do $a; l_{ab}; b$. Their continuations are again p and q, which we assumed bisimilar, so are related by $\mathcal{R}$.

Case $op = (r)-$**:** consider the relation

$$\mathcal{R} = \{((r)p, (r)q) \mid p \sim^{NC} q\} \cup \sim^{NC}$$

We have to show that $\mathcal{R}$ is a bisimulation. Take $((r)p, (r)q) \in \mathcal{R}$ and consider $(r)p \xrightarrow{\alpha} p'$. It is easy to see that $\mathrm{bn}(\alpha) \subseteq \mathrm{bn}(p)$ and, since we assumed $\mathrm{bn}(p) \# q$, we have $\mathrm{bn}(\alpha) \# q$. This transition is inferred either via (RES) or via (OPEN):

- Case (RES): we have $\alpha \equiv_\alpha (r)\alpha'/r$ and the transition is inferred from $p \xrightarrow{\alpha} p'$. Then, by definition of $\mathcal{R}$, there is $q \xrightarrow{\alpha'} q'$, from which (RES) infers $(r)q \xrightarrow{(r)\alpha'/r} q'$. The claim follows from $(p', q') \in \sim^{NC} \subseteq \mathcal{R}$.
- Case (OPEN): we have $\alpha \equiv \bullet; W/r; \overline{a}(r)$ and the transition is inferred from $p \xrightarrow{\bullet; W; \overline{a}r} p'$. By definition of $\mathcal{R}$ there is $q \xrightarrow{\bullet; W; \overline{a}r} q'$, and using (OPEN) we get $(r)q \xrightarrow{\bullet; W/r; \overline{a}(r)} (r)q'$. From $p' \sim^{NC} q'$ it follows $((r)p', (r)q') \in \mathcal{R}$.

A.1.3 Proof of Proposition 3.5.4

We have to prove that

$$\mathcal{R}_{in} = \{(a(r).p, a(r).q) \mid p \approx^{NC} q\} \cup \approx^{NC}$$

is a bisimulation and is closed under renamings. For a pair in $\mathcal{R}_{in}$, we have

$$a(r).p \xrightarrow{a(r')} p[r'/r] \qquad a(r).q \xrightarrow{a(r')} q[r'/r]$$

From $p \approx^{NC} q$ it follows $p[r'/r] \approx^{NC} q[r'/r]$, and also $p[r'/r]\sigma \approx^{NC} q[r'/r]\sigma$, for all σ.

A.2 Proofs for Chapter 4

A.2.1 Proof of Proposition 4.2.2

Let $\mathbf{F}$ the small category of finite ordinals and functions. Being a skeleton of **FinSet**, $\mathbf{F}$ is equivalent to it. Let $F\colon \mathbf{F} \to \mathbf{FinSet}$ be the fully faithful and essentially surjective functor for such equivalence. Then it is easy to check that the functor $F \circ (-)\colon \mathbf{F}^{\rightrightarrows} \to \mathbf{FinSet}^{\rightrightarrows}$ is fully

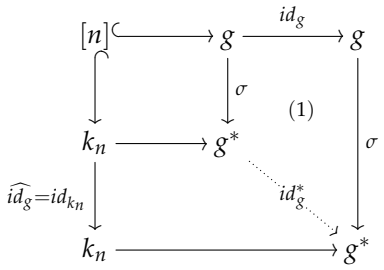
faithful and essentially surjective as well. This means that $\mathbf{F}^{\rightrightarrows}$ and $\mathbf{FinSet}^{\rightrightarrows}$ are equivalent, so $\mathbf{FinSet}^{\rightrightarrows}$ is essentially small and its skeletal category $\mathbf{G}$ is small.

As for finite colimits and pullbacks: they exist in $\mathbf{FinSet}^{\rightrightarrows}$, because it is a *topos* [44], so also in $\mathbf{G}$, by equivalence. Finally, stability of monos under pushouts is a well-known property of topoi.

A.2.2 Proof of Proposition 4.2.3

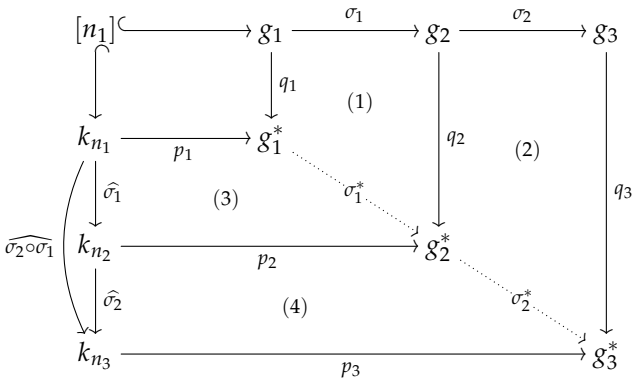
We prove each requirement for the functoriality (up to natural isomorphism):

- $\delta^{\bullet\to\bullet}(id_g) = id_{\delta^{\bullet\to\bullet}(g)}$: by commutativity of (1) in the following diagram



  Notice that the mediating arrow may be any isomorphism i: in this case the vertical leg of the outer pushout would be $i \circ \sigma$. This is not an issue, since functoriality is up to isomorphism.

- $\delta^{\bullet\to\bullet}(\sigma_2 \circ \sigma_1) = \delta^{\bullet\to\bullet}(\sigma_2) \circ \delta^{\bullet\to\bullet}(\sigma_1)$, for any $\sigma_1 \colon g_1 \to g_2$, $\sigma_2 \colon g_2 \to g_3$: by commutativity of



  In particular, we have

$$\begin{aligned} \sigma_2^* \circ \sigma_1^* \circ q_1 &= \sigma_2^* \circ q_2 \circ \sigma_1 && \text{(by commutativity of } (1)\text{, precomposing } \sigma_2^*) \\ &= q_3 \circ \sigma_2 \circ \sigma_1 && \text{(by commutativity of } (2)) \end{aligned}$$

and

$$\begin{aligned}\sigma_2^* \circ \sigma_1^* \circ p_1 &= \sigma_2^* \circ p_2 \circ \widehat{\sigma_1} && \text{(by commutativity of (3), precomposing } \sigma_2^*) \\ &= p_3 \circ \widehat{\sigma_2} \circ \widehat{\sigma_1} && \text{(by commutativity of (4))} \\ &= p_3 \circ \widehat{\sigma_2 \circ \sigma_1}\end{aligned}$$

which means that $\sigma_2^* \circ \sigma_1^*$ commutes with both the smallest and the biggest pushout. By the universal property of the smallest one, this morphism is the unique mediating morphism between them.

A.2.3 Proof of Proposition 4.2.5

It is well known that constant functors and $\mathscr{P}_c$ are accessible and preserve weak pullbacks; $\Delta^\bullet$ and $\Delta^{\bullet\to\bullet}$ have both left and right adjoints, namely functors computing left and right Kan extensions along $\delta^\bullet$ and $\delta^{\bullet\to\bullet}$, so they preserve limits, in particular weak pullbacks, and (filtered) colimits.

A.2.4 Proof of Proposition 4.3.5

Given a coalgebra (P, ρ), the equivalent $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS $(P, \longrightarrow_\rho)$ is given by

$$g \vdash p \xrightarrow{\alpha}_\rho g' \vdash p' \iff (\alpha, p' \in Pg') \in \rho_g(p)$$

In fact, (i) of Definition 4.3.4 reflects the definition of B_n, (ii) the naturality of ρ.

A.2.5 Proof of Proposition 4.3.6

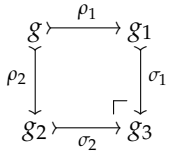
Given a B_n-bisimulation R, we show that $\{Rg\}_{g \in |\mathbf{G}_\mathrm{I}|}$ is a $\mathbf{G}_\mathrm{I}$-indexed bisimulation. By Definition 2.2.15 there is a coalgebra structure map ρ for R, so by Proposition 4.3.5 there is an equivalent $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS $(R, \longrightarrow_\rho)$. Consider $g \in |\mathbf{G}_\mathrm{I}|$ and a a pair $(p, q) \in Rg$. Then there is one transition of the form

$$g \vdash (p, q) \xrightarrow{\alpha}_\rho g' \vdash (p', q') \ , (p', q') \in Rg' \ ,$$

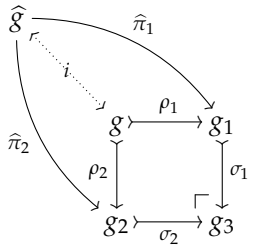
for each transition of p. This gives the simulating transition of q required by (i) of Definition 2.2.19. Condition (ii) just amounts to saying that R is a presheaf $\mathbf{G}_\mathrm{I} \to \mathbf{Set}$.

A.2.6 Proof of Lemma 4.3.8

Let



be any pullback in $\mathbf{G}_{\mathrm{I}}$. It is a pullback also in $\mathbf{FinSet}^{\rightrightarrows}$, therefore there is a unique isomorphism i such that the following diagram in $\mathbf{FinSet}^{\rightrightarrows}$ commutes

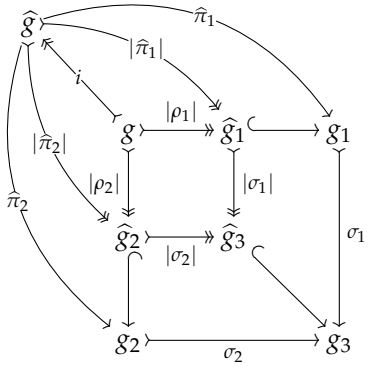


where

$$\begin{aligned}\widehat{g} = \{ \quad & \{(e_1, e_2) \in e_{g_1} \times e_{g_2} \mid \sigma_1(e_1) = \sigma_2(e_2)\}, \\ & \{(v_1, v_2) \in v_{g_1} \times v_{g_2} \mid \sigma_1(v_1) = \sigma_2(v_2)\}, \\ & s_{\widehat{g}} := \lambda(e_1, e_2).(s_{g_1}(e_1), s_{g_2}(e_2)), \\ & t_{\widehat{g}} := \lambda(e_1, e_2).(t_{g_1}(e_1), t_{g_2}(e_2)) \quad \}\end{aligned}$$

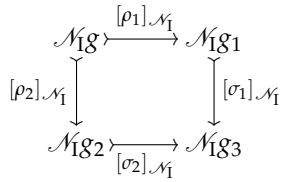
and $\widehat{\pi}_1$, $\widehat{\pi}_2$ are the restrictions to $\widehat{g}$ of the projections from $g_1 \times g_2$.

Let $\widehat{g}_1$, $\widehat{g}_2$ and $\widehat{g}_3$ denote $\widehat{\pi}_1(\widehat{g})$, $\widehat{\pi}_2(\widehat{g})$ and $\sigma_1(\widehat{g}_1)$ $(= \sigma_2(\widehat{g}_2))$, respectively. Then we can decompose each morphism as an isomorphism followed by an embedding as follows



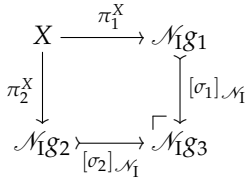
(A.1)

Our goal is showing that the following diagram is a pullback in **Set**



All the legs are indeed injections: $[\rho_i]_{\mathcal{N}}$ and $[\sigma_i]_{\mathcal{N}}$ $(i = 1, 2)$ are injective, because homset functors preserve monomorphisms, thus so are their homomorphic extensions to processes.

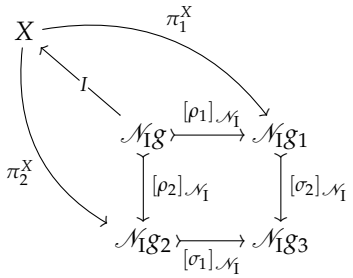
Consider the following pullback



where

$$X = \{(p_1, p_2) \in \mathscr{N}_{\mathrm{I}}g_1 \times \mathscr{N}_{\mathrm{I}}g_2 \mid p_1[\sigma_1]_{\mathscr{N}_{\mathrm{I}}} = p_2[\sigma_2]_{\mathscr{N}_{\mathrm{I}}}\}$$

and π_1^X, π_2^X are the restrictions to X of the projections from $\mathscr{N}_{\mathrm{I}}g_1 \times \mathscr{N}_{\mathrm{I}}g_2$. We shall show that the mediating morphism $I\colon \mathscr{N}_{\mathrm{I}}g \to X$ for the following diagram



is bijective, and thus, since limits are unique up to a unique isomorphism, the square is a pullback.

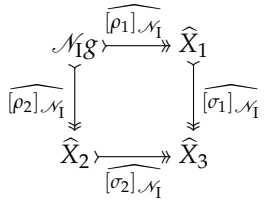
The idea is "lifting" the decomposition of diagram (A.1) to **Set**. First of all, let $\mathcal{N}_F\colon \mathbf{FinSet}^{\rightrightarrows} \to \mathbf{Set}$ be the functor that acts as $\mathcal{N}$ on the whole $\mathbf{FinSet}^{\rightrightarrows}$ and let

$$\widehat{X}_i \;:=\; \{p \in \mathscr{N}_{\mathrm{I}}g_i \mid \mathrm{fn}(p) \subseteq \mathcal{N}_F\widehat{g}_i\} \qquad i = 1, \ldots, 3$$

be the subset of $\mathscr{N}_{\mathrm{I}}g_i$ containing processes with free names in $\mathscr{N}_{\mathrm{I}}\widehat{g}_i$. Notice that $img([\rho_i]_{\mathscr{N}_{\mathrm{I}}}) = \widehat{X}_i$ and that $img([\sigma_i]_{\mathscr{N}_{\mathrm{I}}}|_{\widehat{X}_i}) = \widehat{X}_3$, by commutativity of A.1, so the following functions are properly defined:

$$\widehat{[\rho_i]_{\mathscr{N}_{\mathrm{I}}}} \;:=\; [\rho_i]\colon \mathscr{N}_{\mathrm{I}}g \to \widehat{X}_i \qquad \widehat{[\sigma_i]_{\mathscr{N}_{\mathrm{I}}}} \;:=\; \lambda p \in \widehat{X}_i.p[\sigma_i]\colon \widehat{X}_i \to \widehat{X}_3 \qquad i = 1, 2 \tag{A.2}$$

These are the homomorphic extension to processes of $[|\rho_i|]_{\mathcal{N}_F}$ and $[|\sigma_i|]_{\mathcal{N}_F}$, which are bijections, so are themselves bijective. Their definition ensures the commutativity of



(A.3)

Now we prove that all and only the processes in $\widehat{X}_1$ and $\widehat{X}_2$ occur in a pair in X, and each process appears in only one pair. This will allow us to turn the projections from X into bijective functions by restricting their codomains to $\widehat{X}_1$ and $\widehat{X}_2$.

Formally, we have to show that:

(i) every $(p_1, p_2) \in X$ is such that $p_1 \in \widehat{X}_1$ and $p_2 \in \widehat{X}_2$;

(ii) for all $p_1 \in \widehat{X}_1$ (resp. $p_2 \in \widehat{X}_2$) there is only one $p_2 \in \widehat{X}_2$ (resp. $p_1 \in \widehat{X}_1$) such that $(p_1, p_2) \in X$;

As for (i), let $S_1 = \mathrm{fn}(p_1) \cap (\mathcal{N}_F g_1 \setminus \mathcal{N}_F \widehat{g}_1)$ and suppose that S_1 is not empty. Let $x \in S_1$ and $x' = x[\sigma_1]_{\mathcal{N}_F}$. We have two cases:

1. Every $y \in \mathrm{fn}(p_2)$ is such that $y[\sigma_2]_{\mathcal{N}_F} \neq x'$: then x cannot be in $\mathrm{fn}(p_1)$, because otherwise we would have $p_1[\sigma_1]_{\mathcal{N}_\mathrm{I}} \neq p_2[\sigma_2]_{\mathcal{N}_\mathrm{I}}$;
2. There is $y \in \mathrm{fn}(p_2)$ such that $y[\sigma_2]_{\mathcal{N}_F} = x'$: then x and y stem from items i of g_1 and j of g_2, respectively, such that $\sigma_1(i) = \sigma_2(j)$, so i appears also in $\widehat{g}_1$, which implies $x \in \mathcal{N}_F \widehat{g}_1$.

Both cases imply $x \notin S_1$, which is absurd.

As for (ii), consider the following function

$$\varphi \colon \widehat{X}_1 \xrightarrow{\widehat{[\rho_1]_{\mathcal{N}_\mathrm{I}}}^{-1}} \mathcal{N}_\mathrm{I} g \xrightarrow{\widehat{[\rho_2]_{\mathcal{N}_\mathrm{I}}}} \widehat{X}_2 \ ;$$

we can let p_2 be $\varphi(p_1)$, in fact we have

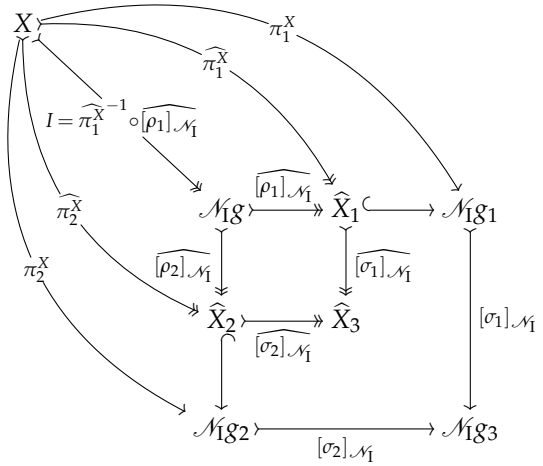
$$\begin{aligned} p_2[\sigma_2]_{\mathcal{N}_\mathrm{I}} &= p_2\widehat{[\sigma_2]_{\mathcal{N}_\mathrm{I}}} && \text{(by } p_2 \in \widehat{X}_2\text{)} \\ &= p_1\widehat{[\sigma_1]_{\mathcal{N}_\mathrm{I}}} && \text{(by commutativity of (A.3))} \\ &= p_1[\sigma_1]_{\mathcal{N}_\mathrm{I}} && \text{(by (A.2))} \end{aligned}$$

so $(p_1, p_2) \in X$. Now, suppose there is another p_2' such that $(p_1, p_2') \in X$. By definition of X, $p_1[\sigma_1]_{\mathcal{N}_\mathrm{I}} = p_2'[\sigma_2]_{\mathcal{N}_\mathrm{I}}$, but also $p_1[\sigma_1]_{\mathcal{N}_\mathrm{I}} = p_2[\sigma_2]_{\mathcal{N}_\mathrm{I}}$, so $p_2[\sigma_2]_{\mathcal{N}_\mathrm{I}} = p_2'[\sigma_2]_{\mathcal{N}_\mathrm{I}}$ and, by injectivity of $[\sigma_2]_{\mathcal{N}_\mathrm{I}}$, $p_2' = p_2$.

Now, thanks to the above observations, we have that the two functions

$$\widehat{\pi_1^X} \colon X \to \widehat{X}_1 \qquad \widehat{\pi_2^X} \colon X \to \widehat{X}_2$$

are well-defined and bijective. Therefore we have that the following diagram commutes



A.2.7 Proof of Theorem 4.3.10

(i) One direction is given by Proposition 4.3.6, the other one by Proposition 4.3.7.

(ii) Given a $\mathbf{G}_\mathrm{I}$-indexed bisimulation $\{R_g\}_{g\in|\mathbf{G}_\mathrm{I}|}$ on $(\mathscr{N}_\mathrm{I}, \longrightarrow_\nu)$, we will show that

$$R^* := \bigcup_{g\in|\mathbf{G}_\mathrm{I}|} R_g$$

is a network conscious bisimulation that is closed under injective renamings. Consider $(p,q) \in R^*$ and suppose $p, q \in \mathscr{N}_\mathrm{I} g$. Condition (i) of Definition 2.2.19 says that p and q are able to simulate each other and their continuations are again in R^*; freshness of $\mathrm{bn}(\alpha)$ w.r.t. q is guaranteed due to $\mathrm{bn}(\alpha)$ being generated by $\delta^\bullet$ or $\delta^{\bullet\to\bullet}$, thus fresh by construction. Finally, closure under injective renamings is guaranteed by (ii) of Definition 2.2.19.

Viceversa, consider a network conscious bisimulation R. Then is easy to see that the following family of relations is a $\mathbf{G}_\mathrm{I}$-indexed bisimulation on $(\mathscr{N}_\mathrm{I}, \longrightarrow_\nu)$:

$$\{R_g := R \cap \mathscr{N}_\mathrm{I} g\}_{g\in|\mathbf{G}_\mathrm{I}|}$$

A.2.8 Proof of Proposition 4.5.2

$\mathscr{E}_R$ and $\mathscr{R}$ are a pair of adjoint functors between accessible categories, hence are accessible themselves. Moreover, they are both right adjoints ($\mathscr{R}$ is right adjoint to the left Kan extension along $\mathbf{G}_\mathrm{I} \hookrightarrow \mathbf{G}$), thus preserve limits. Therefore, being $\widehat{B}_n$ the composition of three accessible and weak-pullback-preserving functors, it has the same properties.

A.2.9 Proof of Proposition 4.5.3

One direction of the mapping is given by

$$(\mathscr{R}P, \rho) \quad\longmapsto\quad (P, \rho') \qquad \rho' : P \xrightarrow{\eta_P} \mathscr{E}_R\mathscr{R}P \xrightarrow{\mathscr{E}_R\rho} \mathscr{E}_R B_n \mathscr{R}P = \widehat{B}_n P$$

the other one by

$$(P, \phi) \quad\longmapsto\quad (\mathscr{R}P, \phi') \qquad \phi' : \mathscr{R}P \xrightarrow{\mathscr{R}\phi} \mathscr{R}\widehat{B}_n P = \mathscr{R}\mathscr{E}_R B_n \mathscr{R}P \xrightarrow{\varepsilon_{B_n\mathscr{R}P}} B_n\mathscr{R}P$$

A.2.10 Proof of Proposition 4.5.5

We show the direction from $\widehat{B}$-coalgebras to $\mathbf{G}_\mathrm{I}$-IL$_\mathrm{nc}$TS$_{SAT}$s, the other direction is analogous. Consider a $\widehat{B}_n$-coalgebra (P, ρ) and the transition relation $\longrightarrow_\rho$ given by

$$g \vdash p \xrightarrow{(\sigma,\alpha)}_\rho g'' \vdash p' \iff (\alpha, p' \in P(g'')) \in (\rho_g(p))_\sigma \ . \tag{A.4}$$

We prove that $(P, \longrightarrow_\rho)$ satisfies all the requirements of Definition 4.5.4. Let t be the tuple $\rho_g(p)$, then we have

$$t \in \prod_{\sigma: g \to g' \in \|\mathbf{G}\|} B_n \mathscr{R} P(g') \ .$$

So, for any transition as in (A.4) with $\sigma\colon g \to g'$, we have $(\alpha, p') \in t_\sigma = B_n \mathscr{R} P(g')$. Then $\alpha \in Lab_{\mathrm{nc}}(g')$ and, since B_n and (ii) of Definition 4.3.4 have been shown to agree, we have that $g'' \vdash p'$ is a valid continuation for α according to (ii) of Definition 4.3.4. This proves (i) of Definition 4.5.4.

As for (iii.a), it follows from the closure under monos condition characterizing the tuples in $\widehat{B}_n P(g)$, namely $t_{\sigma' \circ \sigma} = t_\sigma[\sigma']_{B_n \mathscr{R} P}$. According to (A.4), this equation says that each transition of $g \vdash p$ with σ in its label corresponds to a unique one with $\sigma' \circ \sigma$, and viceversa. This correspondence is as follows. Let (σ, α) and $g'' \vdash p'$ be label and continuation of a transition of $g \vdash p$. Then α and p' agree with (ii) of Definition 4.3.4, as we have already proven, and the result of applying $[\sigma']_{B_n \mathscr{R} P}$ to this pair is described by (ii) of Definition 4.3.4. This indeed gives a path and continuation of the form required by (iii.a).

Finally, (iii.b) reflects the naturality of ρ, explicitly

$$\begin{array}{ccc} p \in Pg & \xmapsto{\rho_g} & t \text{ s.t. } (\alpha, p') \in t_{\sigma' \circ \sigma} \\ {\scriptstyle [\sigma]_P}\downarrow & & \downarrow{\scriptstyle [\sigma]_{\widehat{B}_n P}} \\ p[\sigma]_P \in Pg' & \xmapsto[\rho_{g'}]{} & t' \text{ s.t. } (\alpha, p') \in t'_{\sigma'} \end{array}$$

A.2.11 Proof of Theorem 4.5.7

(i) Analogous to (i) of Theorem 4.3.10.

(ii) Each $\mathbf{G}$-indexed bisimulations $\{R_g\}_{g \in |\mathbf{G}|}$ on $(\mathscr{N}, \longrightarrow_{\nu_{SAT}})$ is equivalent to a $\widehat{B}_n$-bisimulation R, thanks to (i). By Proposition 4.5.3 (instantiated to those B_n-coalgebras that are B_n-bisimulations) R is in turn equivalent to a B_n-bisimulation $\mathscr{R}R$. By Theorem 4.3.10, this corresponds to a network conscious bisimulation, and precisely to $\bigcup_{g \in |\mathbf{G}_\mathrm{I}|} \mathscr{R}R(g)$, which is clearly equal to $\bigcup_{g \in |\mathbf{G}|} R_g$, thus is closed under all renamings.

A.2.12 Proof of Proposition 4.5.1

For the sake of simplicity, we let $J := \mathbf{G}_\mathrm{I} \hookrightarrow \mathbf{G}$ and, since $\mathbf{G}_\mathrm{I}$ and $\mathbf{G}$ have the same objects, we shall use a instead of Ja.

We prove that the formula we have given for $\mathscr{E}_R P(g)$ is correct. Recall from §2.2.3 that $\mathscr{E}_R P(g)$ is defined as the limit of the diagram $P'\colon g \downarrow J \to \mathbf{Set}$, given by

$$\begin{array}{rcl} (\sigma\colon g \to g', g') & \longmapsto & P(g') \\ \rho\colon (\sigma\colon g \to g', g') \to (\sigma'\colon g \to g'', g'') & \longmapsto & \rho\colon g' \to g'' \in \|\mathbf{G}_\mathrm{I}\| \end{array}$$

It is well known that limits can be computed via products and equalizers (see e.g. [41, V,Theorem 2]): given the following morphisms φ and ψ

$$\prod_{a\in|g\downarrow J|} P'a \underset{\psi}{\overset{\varphi}{\rightrightarrows}} \prod_{\rho\in\|g\downarrow J\|} P'(cod(\rho)) \qquad (\varphi(t))_\rho = t_{cod(\rho)} \qquad (\psi(t))_\rho = t_{dom(\rho)}[\rho]_{P'}$$

then the limit of P' is their equalizer, explicitly

$$\{t \in \prod_{a\in|g\downarrow J|} P'a \mid \forall \rho \in \|g\downarrow J\| : t_{cod(\rho)} = t_{dom(\rho)}[\rho]_{P'}\} \ .$$

Expanding the definition of P' in the latter expression

$$\prod_{(\sigma\colon g\to g',g')\in|g\downarrow J|} P'(\sigma\colon g\to g', g') = \prod_{\sigma\colon g\to g'\in\|\mathbf{G}\|} P(g') \qquad t_{dom(\rho)}[\rho]_{P'} = t_{dom(\rho)}[\rho]_P$$

and observing that if ρ has domain σ, then its codomain is $\rho\circ\sigma$ (where this ρ is in $\mathbf{G}$), we get the claim.

The correctness of our formula for $\mathscr{E}_R P(\sigma)$ follows from the fact that the morphisms from the vertex to the base in diagram (2.3) are projections: this turns (2.4) into the visual proof of the claim.

A.3 Proofs for Chapter 5

A.3.1 Proof of Proposition 5.2.5

By induction on the inference of $p \overset{\Lambda}{\Rightarrow} q$. We show the most relevant case, i.e. when this transition is inferred through (COM). We have that $p \overset{\Lambda}{\Rightarrow} q$ is inferred from

$$p \xrightarrow{\Lambda'=(R)(\bullet;W;\vec{a}br \mid ab'x;W';\bullet \mid \Theta)} q' \ ,$$

so $\Lambda \equiv_\Lambda (R')(\bullet; W; W'; \bullet \mid \Theta)$ and $q \equiv (R'')q'(\sigma_b \circ \sigma_r)$. By induction hypothesis Λ' and q' are well-formed, so also Λ is well-formed, because it is obtained from Λ' just by removing some names, and it is simple, otherwise (COM) could not be applied.

We prove that $(R'')q'(\sigma_b \circ \sigma_r)$ is well-formed. First, notice that $q'(\sigma_b \circ \sigma_r)$ is clearly well-formed, because σ_b and σ_r avoid captures. Now we show that, if $a \in R''$, then $l_{ab} \in \mathrm{fn}(q'(\sigma_b \circ \sigma_r))$ only if $l_{ab} \in R''$. Suppose, by absurd, that $l_{ab} \in \mathrm{fn}(q'(\sigma_b \circ \sigma_r))$ but $l_{ab} \notin R''$. Then we have that l_{ab} is either in $\mathrm{fn}(p)$ or in $\mathrm{Bn}(\Lambda) \cup \mathrm{Obj}_{\mathrm{in}}(\Lambda)$, because this set contains the new names in the continuation, i.e. those not in $\mathrm{fn}(p)$. By cases:

- $l_{ab} \in \mathrm{fn}(p)$ is forbidden due to $a \in R'' \subseteq \mathrm{bn}(p)$, because we assumed that free and bound names of p are distinct.
- if $l_{ab} \in \mathrm{Bn}(\Lambda)$ then either $l_{ab} \in R'$ or $(l_{(ab)})$ is the placeholder of an input path in Λ. In the first case also a must be in R', because $l_{ab} \in \mathrm{Obj}(\Theta)$ and thus $a \in \mathrm{Obj}(\Theta)$, but then

a cannot be in R'', as it is defined so that $R'' \cap R' = \varnothing$; this is a contradiction. In the second case, a would be already bound in Λ, but this is not allowed by the inference rules.

- $l_{ab} \in \mathrm{Obj_{in}}(\Lambda)$ is forbidden due to $a \in R$: in fact, if l_{ab} were free in Λ, (a) would capture it, causing Λ not to be well-formed, against the inductive hypothesis.

Summarizing, either $l_{ab} \notin \mathrm{fn}(q'(\sigma_b \circ \sigma_r))$ or $l_{ab} \in \mathrm{fn}(q'(\sigma_b \circ \sigma_r))$ and $l_{ab} \in R''$, so q is well-formed.

A.3.2 Proof of Theorem 5.2.7

We introduce the following notation:

$\Pi, \Pi', \ldots, \Pi_1, \Pi_2, \cdots$:	(forrests of) proof trees
$\Pi\sigma$:	proof obtained by applying σ to every process and concurrent path in Π
$\mathrm{dom}(\sigma)$:	set of names that are not mapped to themselves by σ;
$\mathrm{img}(\sigma)$:	image of $\mathrm{dom}(\sigma)$ through σ;
$\mathrm{ds}(\alpha)$:	destination site of a path α, if any.
$x\sigma^*$:	result of applying σ to both the free and bound names of x

We call *non-linear rules* those rules that depend on the equality of certain names in the premises, namely (COM), (SRV-IN), (SRV-OUT), (SRV-SRV), which, in fact, can only be applied to paths with matching interaction sites. We call all the other ones *linear rules*.

We need a plethora of lemmata in order to prove the main result.

Lemma A.3.1. *Suppose $p \overset{\Lambda}{\Rightarrow} q$ and let Ω be a sequence of non-linear rule instances inferring $p \overset{\Lambda'}{\Rightarrow} q'$ from this transition. Then q' is of the form $(R)q\sigma$, for some R and σ. Moreover, given any other transition $p'' \overset{\Lambda}{\Rightarrow} q''$, there is a sequence of non-linear rule instances that infers $p'' \overset{\Lambda'}{\Rightarrow} (R)q''\sigma$ from it.*

Proof. Non-linear rules, and in particular (COM), can only modify the continuation of the premise by adding restrictions and renaming it. This justifies the first part of the statement. The second part follows from observing that each rule instance in Ω does not look at the source process, so we can replace p with p'' and q with q'' in Ω in order to infer $p'' \overset{\Lambda'}{\Rightarrow} (R)q''\sigma$ from $p'' \overset{\Lambda}{\Rightarrow} q''$. □

Lemma A.3.2. *We can define three operations on proofs:*

(i) **renaming:** *Given $p \overset{\Lambda}{\Rightarrow} q$, with proof Π, and a renaming σ such that $\Lambda\sigma$ is simple, then $p\sigma \overset{\Lambda\sigma}{\Longrightarrow} q\sigma$ has proof $\Pi\sigma$.*

(ii) **α-conversion**: *Given $p \overset{\Lambda}{\Rightarrow} q$, with proof Π, for any $\sigma = [r'/r]$ such that $r \in \mathrm{Bn}(\Lambda)$ and r' is fresh w.r.t. p and Λ, we have that $p\sigma^* \overset{\Lambda\sigma^*}{\Longrightarrow} q\sigma$ has proof $\Pi\sigma^*$.*

(iii) **Input object restriction**: *Given* $p \xrightarrow{(R)(abr;W;\bullet \mid \Theta)} q$, *with proof* Π, *suppose* $a(s).p' \overset{aar;\bullet}{\Longrightarrow} p'[r/s]$ *is the axiom for the input path in* Π. *Then, for any fresh* s', *there is a transition*

$$p[s'/s]^* \xrightarrow{(R)(ab(s');W;\bullet \mid \Theta)} q'$$

whose proof is obtained from Π *as follows:*

(a) r *is replaced with* (s') *in each input path from which* $abr; W; \bullet$ *is constructed;*

(b) $p'[r/s]$ *is replaced with* $p'[s'/s]$ *throughout the proof.*

Moreover, $q'[r/s'] = q$.

Proof. All the three statements can be proved by induction on the depth of Π.

(i) By cases on the type of the last rule applied in Π; we show some relevant ones:

Case (OPEN): Then $p = (r'')p'$, $\Lambda \equiv_\Lambda (r'')(\Lambda' /\!\!/ r'')$ and the transition is inferred from $p \overset{\Lambda'}{\Longrightarrow} q$. Suppose this last transition has proof Π'. $\Lambda'\sigma$ is simple because, if r'' is a link, σ does not map any other link to it by Convention A.1, so the induction hypothesis can be applied, yielding $p'\sigma \overset{\Lambda'\sigma}{\Longrightarrow} q\sigma$ with proof $\Pi'\sigma$. This transition is a valid premis for (OPEN) because σ, according to Convention A.1, does not replace any free name with r''. The application of (OPEN) gives the required transition and, together with $\Pi'\sigma$, yields the required proof.

Case (PAR): Then $p = p_1 \mid p_2$, $\Lambda \equiv_\Lambda \Lambda_1 \mid \Lambda_2$ and the transition is inferred from $p_1 \overset{\Lambda_1}{\Longrightarrow} q_1$ and $p_2 \overset{\Lambda_2}{\Longrightarrow} q_2$, so $q = q_1 \mid q_2$. Suppose these transitions have proofs Π_1 and Π_2. The fact that $(\Lambda_1 \mid \Lambda_2)\sigma$ is simple clearly implies that so are $\Lambda_1\sigma$ and $\Lambda_2\sigma$. Therefore, by induction hypothesis, $p_1\sigma \overset{\Lambda_1\sigma}{\Longrightarrow} q_1\sigma$ and $p_2\sigma \overset{\Lambda_2\sigma}{\Longrightarrow} q_2\sigma$ have proofs $\Pi_1\sigma$ and $\Pi_2\sigma$, respectively. Moreover, by Convention A.1, σ cannot break (PAR) side conditions, so an application of (PAR) gives the thesis.

Case (COM): Then the transition is inferred from $p \xrightarrow{(R)(\alpha_1 \mid \alpha_2 \mid \Theta)} q$, with $\alpha_1 = \bullet; W; \overline{a}br$ and $\alpha_2 = ab'x; W'; \bullet$, so $\Lambda \equiv_\Lambda (R')(\bullet; W; W'; \bullet \mid \Theta)$ and $q = (R'')q'\sigma'$. Suppose the premise has proof Π'. By Convention A.1, bound names in the label are not affected by σ, so $\Lambda\sigma \equiv_\Lambda (R')(\bullet; W\sigma; W'\sigma; \bullet \mid \Theta\sigma)$. This concurrent path is simple, therefore also $\alpha_1\sigma$ and $\alpha_2\sigma$ are simple. Then, by induction hypothesis, we have $p\sigma \xrightarrow{(R)(\alpha_1\sigma \mid \alpha_2\sigma \mid \Theta\sigma)} q'\sigma$ with proof $\Pi'\sigma$. We can conclude by applying (COM) to this transition, which yields

$$p\sigma \xrightarrow{(R')(\bullet;W\sigma;W'\sigma;\bullet \mid \Theta\sigma)} (R'')q'(\sigma' \circ \sigma) = ((R'')q'\sigma')\sigma \ ,$$

where σ can be brought outside because σ' is a map between bound names of p, and Convention A.1 tells us that these names are not affected by σ.

(ii) A straightforward adaptation of the proof above. The cases (RES) and (OPEN) are treated by applying the renaming operation to the proof of the premise, with $\sigma = [r'/r]$ whenever $p = (r)p'$. Notice that σ^* cannot make paths non-simple, because r and r' are both fresh w.r.t. the links appearing in the involved labels.

(iii) By cases on the type of the last applied in Π. We show two cases: the first clarifies the most, the second exemplifies the inductive step.

Case (IN): Then $p = a(s).p'$, Λ is just $aar;\bullet$, $q = p'[r/s]$ and Π is an axiom. Then Π' is

$$a(s').p'[s'/s] \xrightarrow{aa(s);\bullet} p'[s'/s]$$

which clearly satisfies (a) and (b). Finally, we have

$$(p'[s'/s])[r/s'] = p'[r/s] \ .$$

Case (OPEN): Then $p = (r')p'$, $\Lambda \equiv_\Lambda (r')(R)(abr; W; \bullet /\!\!/ r' \mid \Theta /\!\!/ r')$ and the transition is inferred from $p' \xrightarrow{(R)(abr;W;\bullet \mid \Theta)} q$. The proof for this transition clearly contains the same input axiom for $abr; W; \bullet$ as Π so, by induction hypothesis, there is

$$p'[s'/s]^* \xrightarrow{(R)(ab(s');W;\bullet \mid \Theta)} q'$$

with a proof obtained according to (a) and (b). We get the required proof by applying (OPEN) to this transition.

□

Lemma A.3.3. *Suppose $p \overset{\Lambda}{\Rightarrow} q$ has a proof with the following structure*

$$\dfrac{\begin{array}{c}\Pi \\ \textit{non-linear rule } \rho_1 \\ \textit{linear rule } \rho_2\end{array}}{p \overset{\Lambda}{\Rightarrow} q}$$

and let α_1 and α_2 be the paths concatenated by ρ_1. Suppose $\mathrm{is}(\alpha_1) \cap \mathrm{is}(\alpha_2) \cap \mathrm{bn}(p) = \varnothing$. Then there are: a proof Π', a linear rule instance ρ_2', a non-linear rule instance ρ_1', a process $p^* \equiv p$ and another process q^*, which is obtained by bringing some unguarded restrictions of q to the top level, such that the following is a valid proof

$$\dfrac{\begin{array}{c}\Pi' \\ \rho_2' \\ \rho_1'\end{array}}{p^* \overset{\Lambda}{\Rightarrow} q^*}$$

Proof. We have to consider all possible pairs of ρ_1 and ρ_2 such that the objects of α_1 and α_2 are involved in the side conditions of ρ_2. If they are not, we can simply put $\rho_1' = \rho_2$, $\rho_2' = \rho_1$ and $\Pi' = \Pi$. The only non-trivial cases are when ρ_2 is (RES), (OPEN) or (PAR). We show the proof of the statement for ρ_1 being (COM), which is the most involved case.

Case $\rho_2 =$ (RES): Then $p = (r')p'$ and the transition is inferred as follows

$$\text{(RES)}\ \frac{\text{(COM)}\ \dfrac{\Pi}{p' \xrightarrow{(R)(\bullet;W;\vec{a}br \mid ab'x;W';\bullet \mid \Theta)} q'}}{(r')p' \xrightarrow{\Lambda\equiv_\Lambda (R')(\bullet;W/r';W'/r';\bullet \mid \Theta /\!\!/ r')} (r')(R'')q'(\sigma_b \circ \sigma_r)} \qquad \frac{\Pi}{p' \xrightarrow{(R')(\bullet;W;W';\bullet \mid \Theta)} (R'')q'(\sigma_b \circ \sigma_r)}$$

We want (COM) to appear after the rule treating the restriction of r'. We have two cases:

- $r' \in \mathrm{n}(r)$: if $x = r$ then the earlier restriction of r' should be treated through (OPEN). However, we need to make the input object in the top transition bound in order to get a proper premise of (OPEN). Let s be the input placeholder in p' for $ab'x; W'; \bullet$ and let s' be fresh. Then we can apply (iii) of Lemma A.3.2 and get

  $$p'[s'/s]^* \xrightarrow{(R)(\bullet;W;\vec{a}br \mid ab'(s');W';\bullet \mid \Theta)} q'';$$

  where $q''[r/s'] = q'$. Let Π' be the proof of this transition We can put $\rho_2' =$ (OPEN), because (OPEN) side condition is satisfied due to (RES)'s one and the hypothesis $r' \neq a$, and $\rho_1' =$ (COM). So we have

  $$\text{(COM)}\ \frac{\text{(OPEN)}\ \dfrac{\Pi'}{p'[s'/s]^* \xrightarrow{(R)(\bullet;W;\vec{a}br \mid ab'(s');W';\bullet \mid \Theta)} q''}}{(r')p'[s'/s]^* \xrightarrow{(R')(\bullet;W/r';W'/r';\bullet \mid \Theta /\!\!/ r')} (r')(R'')q''([r/s'] \circ \sigma_b)} \qquad \frac{\Pi'}{(r')p'[s'/s]^* \xrightarrow{(r')(R)(\bullet;W/r';\vec{a}br \mid ab'(s');W'/r';\bullet \mid \Theta /\!\!/ r')} q''}$$

  Due to the freshness of s', we can commute the two renamings in the continuation and get $(r')(R'')q'\sigma_b$. This is equal to the continuation of the original proof, as $x = r$ implies $\sigma_r = id$. Finally, the case $x = (s)$ is similar: again we have $\rho_2' =$ (OPEN) adding a restriction (r') and $\rho_1' =$ (COM) closing its scope; the remaining proof is still Π.

- $r' = b \notin \mathrm{n}(r)$: in this case necessarily $b \notin R$, so $b = b'$ and $\sigma_b = id$. We simply have $\Pi' = \Pi$, $\rho_2' =$ (COM) and $\rho_1' =$ (OPEN), which restricts b in the label.

Case $\rho_2 =$ (OPEN): Then $p = (r')p'$ and the transition is inferred as follows

$$\dfrac{\dfrac{\dfrac{\Pi}{p' \xrightarrow{(R)(\bullet;W;\overline{a}br \mid ab'x;W';\bullet \mid \Theta)} q'}\,(\text{COM})}{p' \xrightarrow{(R')(\bullet;W;W';\bullet \mid \Theta)} (R'')q'(\sigma_b \circ \sigma_r)}\,(\text{OPEN})}{(r')p' \xrightarrow{\Lambda \equiv_\Lambda (r')(R')(\bullet;W/r';W'/r';\bullet \mid \Theta /\!\!/ r')} (R'')q'(\sigma_b \circ \sigma_r)}$$

Then we have $r' \in \mathrm{Obj}(\Theta)$, by (OPEN) side condition. The proof of the previous case still applies, with the only exception that the scope of r' is never closed by $\rho_1' = (\text{COM})$, i.e. r' remains restricted in the label because $r' \in \mathrm{Obj}(\Theta)$.

Case $\rho_2 = (\textsc{par})$: Then $p = p_1 \mid p_2$ and the transition is inferred as follows

$$\dfrac{\dfrac{\dfrac{\Pi_1}{p_1 \xrightarrow{(R)(\bullet;W;\overline{a}br \mid ab'x;W';\bullet \mid \Theta)} q_1}\,(\text{COM})}{p_1 \xrightarrow{(R')(\bullet;W;W';\bullet \mid \Theta)} (R'')q_1\sigma} \quad \dfrac{\Pi_2}{p_2 \overset{\Lambda_2}{\Longrightarrow} q_2}\,(\text{PAR})}{p_1 \mid p_2 \xrightarrow{(R')(\bullet;W;W';\bullet \mid \Theta) \mid \Lambda_2} (R'')q_1\sigma \mid q_2}$$

Now, we would like to permute (COM) and (PAR), but some of the bound names that (COM) removes from the label of p_1 may conflict with Λ_2, breaking (PAR) side conditions. So we have to repeatedly apply (ii) of Lemma A.3.2 to α-convert Π_1 w.r.t. those names. Let σ_α^* be the α-converting renaming. We can build the following proof

$$\dfrac{\dfrac{\dfrac{\Pi_1\sigma_\alpha^*}{p_1\sigma_\alpha^* \xrightarrow{(R\sigma_\alpha^*)(\bullet;W;\overline{a\sigma_\alpha}b\sigma_\alpha r\sigma_\alpha \mid a\sigma_\alpha b'\sigma_\alpha x\sigma_\alpha;W';\bullet \mid \Theta)} q_1\sigma_\alpha} \quad \dfrac{\Pi_2}{p_2 \overset{\Lambda_2}{\Longrightarrow} q_2}\,(\text{PAR})}{p_1\sigma_\alpha^* \mid p_2 \xrightarrow{(R\sigma_\alpha^*)(\bullet;W;\overline{a\sigma_\alpha}b\sigma_\alpha r\sigma_\alpha \mid a\sigma_\alpha b'\sigma_\alpha x\sigma_\alpha;W';\bullet \mid \Theta) \mid \Lambda_2} q_1\sigma_\alpha \mid q_2}\,(\text{COM})}{p_1\sigma_\alpha^* \mid p_2 \xrightarrow{(R')(\bullet;W;W';\bullet \mid \Theta) \mid \Lambda_2} (R''\sigma_\alpha^*)(q_1\sigma_\alpha \mid q_2)\sigma'}$$

Notice that σ_α^* affects neither on W, W', as they contain only free names, nor Θ, as its bound names are still in the consequence of (COM) in the original proof. We get the claim thanks to the following:

$$\begin{aligned}
p_1\sigma_\alpha^* \mid p_2 &\equiv p_1 \mid p_2 && (\alpha\text{-conversion}) \\
(R''\sigma_\alpha)(q_1\sigma_\alpha \mid q_2)\sigma' &= (R''\sigma_\alpha)(q_1(\sigma' \circ \sigma_\alpha) \mid q_2) && (\text{freshness of } \mathrm{dom}(\sigma') \text{ w.r.t. } q_2) \\
&\equiv (R'')(q_1\sigma \mid q_2) && (\alpha\text{-conversion})
\end{aligned}$$

where the last congruence holds because σ' maps the α-coverted extruded names to the α-converted input placeholders. Notice that the scope of R'' now includes both $q_1\sigma$ and q_2.

□

Lemma A.3.4. *Given a process p and a renaming σ, consider any $p\sigma \overset{\Lambda}{\Rightarrow} q$ with proof Π. If every application of a non-linear rule in Π concatenates paths α_1 and α_2 such that*

$$\begin{gathered}\mathrm{img}(\sigma)\\ \cap\\ \mathrm{obj}(\alpha_1) \cup (\mathrm{is}(\alpha_1) \cap \mathrm{is}(\alpha_2)) = \varnothing\ ,\end{gathered}$$

then $p \overset{\Lambda'}{\Rightarrow} q'$, where $\Lambda = \Lambda'\sigma$ and $q = q'\sigma$.

Proof. By induction on the depth of Π. Suppose Π has depth at least one (the base cases are trivial), the proof proceeds by cases on the last rule of Π. We show two of them, the other ones are analogous:

Case (OPEN): Then $p = (r'')p'$, $\Lambda \equiv_\Lambda (r'')(\Lambda'' /\!\!/ r)$ and the transition is inferred as follows

$$(\text{OPEN})\ \frac{\begin{matrix}\Pi'\\ p'\sigma \overset{\Lambda''}{\Longrightarrow} q\end{matrix}}{(r'')p'\sigma \overset{(r'')(\Lambda'' /\!\!/ r)}{\Longrightarrow} q}$$

where we replaced $((r'')p')\sigma$ with $(r'')p'\sigma$ thanks to Convention A.1. By induction hypothesis we have $p' \overset{\widetilde{\Lambda}}{\Rightarrow} \widetilde{q}$, where $\Lambda'' = \widetilde{\Lambda}\sigma$ and $q = \widetilde{q}\sigma$. We can apply (OPEN) to this transition because σ does not affect r'', an thus the rule's side condition. This yields $(r'')p' \overset{(r'')(\widetilde{\Lambda} /\!\!/ r'')}{\Longrightarrow} \widetilde{q}$, which satisfies the statement.

Case (COM): Then $\Lambda \equiv_\Lambda (R')\ (\bullet; W; W'; \bullet \mid \Theta)$, $q = (R'')q'\sigma$ and the transition is inferred as follows

$$(\text{COM})\ \frac{\begin{matrix}\Pi'\\ p\sigma \overset{(R)\ (\bullet;W;\overline{a}br'' \mid ab'x;W';\bullet \mid \Theta)}{\Longrightarrow} q'\end{matrix}}{p\sigma \overset{(R')\ (\bullet;W;W';\bullet \mid \Theta)}{\Longrightarrow} (R'')q'\sigma'}$$

By induction hypothesis we have $p \overset{(\widetilde{R})\ (\widetilde{\alpha}_1 \mid \widetilde{\alpha}_2 \mid \widetilde{\Theta})}{\Longrightarrow} \widetilde{q}$ where $\widetilde{\alpha}_1\sigma = \bullet; W; \overline{a}br''$, $\widetilde{\alpha}_2\sigma = ab'x; W'; \bullet$, $\widetilde{R}\sigma = R$, $\widetilde{\Theta}\sigma = \Theta$ and $\widetilde{q}\sigma = q'$. By the hypothesis of our claim, σ acts as the identity on a, b and r'', so we have: $\mathrm{is}(\widetilde{\alpha}_1) = \mathrm{is}(\widetilde{\alpha}_2) = a$; $\mathrm{obj}(\widetilde{\alpha}_1) = \mathrm{obj}(\widetilde{\alpha}_2)$ whenever $r'' \notin R$; $\mathrm{ds}(\widetilde{\alpha}_1) = \mathrm{ds}(\widetilde{\alpha}_2) = b$ whenever $b \notin R$. Therefore we can apply (COM) again and get the desired transition. The continuation of such transition is indeed renamed through the same σ' as the original proof, because extruded names and input placeholders are not affected by σ, by Convention A.1.

□

Lemma A.3.5. *Given a process p and a renaming σ, suppose $p\sigma \stackrel{\Lambda}{\Rightarrow} q$ has proof Π. Then there is a transition $p \stackrel{\Lambda'}{\Rightarrow} q'$ and a sequence Ω of non-linear rule instances such that Ω infers $p\sigma \stackrel{\Lambda}{\Rightarrow} q^*$ from $p\sigma \stackrel{\Lambda'\sigma}{\Longrightarrow} q'\sigma$, where q^* is q with some unguarded restrictions brought to the top level.*

Proof. Since, by Convention A.1, $\mathrm{img}(\sigma) \cap \mathrm{bn}(p) = \varnothing$, we can repeatedly use Lemma A.3.3 to push towards the root of the proof all the instances of non-linear rules in Π that concatenate α_1 and α_2 such that $\mathrm{obj}(\alpha_1)$ or $\mathrm{is}(\alpha_1) \cap \mathrm{is}(\alpha_2)$ are in the image of σ, i.e. those that violate the hypothesis of Lemma A.3.4. More precisely:

1. Take the deepest subproof $\widetilde{\Pi}$ of Π such that Lemma A.3.3 can be applied: let $\widetilde{p} \stackrel{\widetilde{\Lambda}}{\Rightarrow} \widetilde{q}$ be its root transition.
2. Permute the last two rules in $\widetilde{\Pi}$ using Lemma A.3.3: let $\widetilde{\Pi}'$ the resulting proof and $\widetilde{p}^* \stackrel{\widetilde{\Lambda}}{\Rightarrow} \widetilde{q}^*$ its root transition.
3. Replace $\widetilde{\Pi}$ with $\widetilde{\Pi}'$ in Π, and replace $\widetilde{p}$ with $\widetilde{q}^*$ and $\widetilde{q}$ with $\widetilde{q}^*$ in the piece of proof from the root of $\widetilde{\Pi}$ to the root of Π. The resulting proof is indeed valid, because step 2 does not affect the label.

Repeat these steps until obtaining a new proof for $p^*\sigma \stackrel{\Lambda}{\Rightarrow} q^*$. This proof has an upper part Π'' and a lower part Ω', the latter containing all the non-linear rules that violate the hypothesis of Lemma A.3.4 when considering the renaming σ. Now, suppose Π'' has root $p^*\sigma \stackrel{\widehat{\Lambda}}{\Rightarrow} \widehat{q}$. We can apply Lemma A.3.4 to this transition and get the required $p^*\sigma(\equiv p) \stackrel{\Lambda'}{\Rightarrow} q'$ such that $\widehat{\Lambda} = \Lambda'\sigma$ and $\widehat{q} = q\sigma$. Finally, we have to use Lemma A.3.1 to recover Ω from Ω', as they infer transitions with different source processes. □

Lemma A.3.6. *Let $\mathcal{R}$ be the transitive closure of $\bigcup_{n\in\omega} \mathcal{R}_n$, where*

$$\begin{aligned}
\mathcal{R}^1 &= \{((r)p_1 \mid p_2, (r)(p_1 \mid p_2)) \mid r \notin \mathrm{fn}(p_2)\} \\
\mathcal{R}^2 &= \{(p_1 \mid p_2, p_2 \mid p_1)\} \\
\mathcal{R}^3_{n+1} &= \{((r)p, (r)q) \mid (p,q) \in \mathcal{R}_n\} \\
\mathcal{R}^4_{n+1} &= \{(p\sigma, q\sigma) \mid (p,q) \in \mathcal{R}_n\}
\end{aligned}$$

$$\mathcal{R}_0 = \sim^{NC}_{\kappa} \cup \mathcal{R}^1 \cup \mathcal{R}^2 \qquad \mathcal{R}_{n+1} = \mathcal{R}^3_{n+1} \cup \mathcal{R}^4_{n+1} \cup \mathcal{R}_n$$

Then $\mathcal{R}$ is a concurrent network conscious bisimulation.

Proof. Given $(p,q) \in \mathcal{R}$, we have to prove that p and q can simulate each other's transitions and that their continuations are related by $\mathcal{R}$. To do this, we proceed by induction on n, considering any $p' \equiv p$ and $q' \equiv q$ such that $(p',q') \in \mathcal{R}_n$. It is enough to prove that $\bigcup_{n\in\omega} \mathcal{R}_n$ is a bisimulation, because the transitive closure of a bisimulation is clearly a bisimulation as well. For the base case, we treat separately the case of (p',q') being in $\mathcal{R}^1$ and $\mathcal{R}^2$. For the

inductive step, we do the same, considering R^3_{n+1} and R^4_{n+1}, for $n \geq 0$. Considering processes that are structurally congruent to the original ones is harmless, because $\equiv \subseteq \sim_\kappa^{NC} \subseteq \mathcal{R}$, and allows us to ignore transitions that are not inferred directly through the rules.

$\boxed{p \equiv (r)p_1 \mid p_2 \, , \, q \equiv (r)(p_1 \mid p_2)}$ Suppose $((r)p_1 \mid p_2, (r)(p_1 \mid p_2)) \in \mathcal{R}^1$ and consider a transition $(r)p_1 \mid p_2 \stackrel{\Lambda}{\Rightarrow} p'$. We prove that $(r)(p_1 \mid p_2)$ can simulate this transition by cases on the last rule used to infer it.

Case (PAR): Suppose it is inferred as follows

$$
\text{(OPEN)}\ \frac{p_1 \stackrel{\Lambda_1}{\Longrightarrow} q_1}{(r)p_1 \stackrel{(r)(\Lambda_1 /\!\!/ r)}{\Longrightarrow} p_1'}
$$

$$
\vdots
$$

$$
\Omega
$$

$$
\vdots
$$

$$
(r)p_1 \stackrel{\Lambda_1'}{\Longrightarrow} (R)p_1'\sigma
$$

$$
\vdots
$$

$$
\Omega_{(r)}
$$

$$
\vdots
$$

$$
\text{(PAR)}\ \frac{(r)p_1 \stackrel{\Lambda_1''}{\Longrightarrow} (R')(R)p_1'(\sigma_{(r)} \circ \sigma) \qquad p_2 \stackrel{\Lambda_2}{\Longrightarrow} p_2'}{(r)p_1 \mid p_2 \stackrel{\Lambda_1'' \mid \Lambda_2}{\Longrightarrow} (R')(R)p_1'(\sigma_{(r)} \circ \sigma) \mid p_2'}
$$

where Ω and $\Omega_{(r)}$ are sequences of non-linear rule instances (the rule on top may be (RES): this case is analogous). In particular, suppose $\Omega_{(r)}$ applies those rules that treat the communication of (r). These might be zero or many, for instance whenever r is extruded by more than one output path.

The idea is to build a new proof where (PAR) is applied before (OPEN). However, some names in $(\mathrm{Bn}(\Lambda_1) \cup \{r\}) \setminus \mathrm{Bn}(\Lambda_1'')$ that are also in $\mathrm{Fn}(\Lambda_2)$ may prevent the application of these rules. So we use (i) and (ii) of Lemma A.3.2 to replace those names with fresh ones. Let $\widehat{\sigma}$ be the mapping that performs this operation and let $\widehat{r} = r\widehat{\sigma}$. The new proof is

$$
\begin{array}{c}
(\text{PAR})\ \dfrac{p_1\widehat{\sigma}^* \xrightarrow{\Lambda_1\widehat{\sigma}^*} p_1'\widehat{\sigma} \qquad p_2 \xrightarrow{\Lambda_2} p_2}{p_1\widehat{\sigma}^* \mid p_2 \xrightarrow{\Lambda_1\widehat{\sigma}^* \mid \Lambda_2} p_1'\widehat{\sigma} \mid p_2'} \\
\vdots \\
\Omega' \\
\vdots \\
(\text{OPEN})\ \dfrac{p_1\widehat{\sigma}^* \mid p_2 \xrightarrow{\widehat{\Lambda}_1 \mid \Lambda_2} (R\widehat{\sigma})(p_1'\widehat{\sigma} \mid p_2')\sigma'}{(\widehat{r})(p_1\widehat{\sigma}^* \mid p_2) \xrightarrow{(\widehat{r})(\widehat{\Lambda}_1 /\!\!/ \widehat{r} \mid \Lambda_2)} (R\widehat{\sigma})(p_1'\widehat{\sigma} \mid p_2')\sigma'} \\
\vdots \\
\Omega'_{(\widehat{r})} \\
\vdots \\
(\widehat{r})(p_1\widehat{\sigma}^* \mid p_2) \xrightarrow{\Lambda_1'' \mid \Lambda_2} (R'\widehat{\sigma})(R\widehat{\sigma})(p_1'\widehat{\sigma} \mid p_2')\sigma_{(\widehat{r})} \circ \sigma'
\end{array}
$$

This proof is structured as follows. After (PAR), we have a sequence Ω' applying the same kind of rules as Ω to the same pairs of paths (differences in the involved processes are immaterial), except that some extruded names R may be affected by $\widehat{\sigma}^*$: this is why we have $(R\widehat{\sigma})$ in the inferred continuation. The concurrent path $\widehat{\Lambda}_1$ in the label inferred by Ω' is "almost" $\Lambda_1'\widehat{\sigma}^*$: it lacks $(\widehat{r})$, which in fact is added by (OPEN). Communications of this name are derived by $\Omega_{(\widehat{r})}$: it mimics $\Omega_{(r)}$ as Ω' does with Ω. In the end we recover Λ_1'', because $\widehat{\sigma}^*$ was defined to act only on names that disappear throughout the proof due to synchronizations, hence those not in Λ_1''. Among these, we have names extruded by $\Omega_{(r)}$, which in fact appear in the inferred continuation as $R'\widehat{\sigma}$.

Now we have to check that processes in the two proofs match. By α-conversion we have $(\widehat{r})(p_1\widehat{\sigma}^* \mid p_2) \equiv (r)(p_1 \mid p_2)$ and

$$
\begin{aligned}
(R'\widehat{\sigma})(R\widehat{\sigma})(p_1'\widehat{\sigma} \mid p_2')\sigma_{(\widehat{r})} \circ \sigma' &= (R'\widehat{\sigma})(R\widehat{\sigma})(p_1'(\sigma_{(\widehat{r})} \circ \sigma' \circ \widehat{\sigma}) \mid p_2') \\
&\equiv (R')(R)(p_1'(\sigma_{(r)} \circ \sigma) \mid p_2')
\end{aligned}
$$

where the first equation holds due to freshness of $\mathrm{dom}(\sigma_{(\widehat{r})} \circ \sigma')$ w.r.t. p_2' and the last congruence holds because $\widehat{\sigma}$ only affects names that are either being extruded or bound input placeholders in Λ_1, so $\sigma'_{(r)} \circ \sigma' \circ \widehat{\sigma}$ replaces the α-converted input placeholders with α-converted extruded names, but these are again bound by $(R\widehat{\sigma})$ and $(R'\widehat{\sigma})$ in the continuation. The overall effect is the α-conversion of $(R')(R)(p_1'(\sigma_{(r)} \circ \sigma) \mid p_2')$.

Finally, since $\mathcal{R}$ is closed under α-conversion and scope extension, we have

$$
(\ (R'\widehat{\sigma})(R\widehat{\sigma})(p_1'\widehat{\sigma} \mid p_2')\sigma_{(\widehat{r})} \circ \sigma', (R')(R)p_1'(\sigma_{(r)} \circ \sigma) \mid p_2'\ ) \in \mathcal{R}\ .
$$

Case non-linear rule: Suppose the last part of the proof of $(r)p_1|p_2 \stackrel{\Lambda}{\Rightarrow} p'$ is a sequence of non-linear rule applications Ω with an occurrence of (PAR) on top. By the previous case, the transition of $(r)p_1 \mid p_2 \stackrel{\Lambda'}{\Rightarrow} \widetilde{q}$ inferred by (PAR) can be simulated by $(r)(p_1 \mid p_2) \stackrel{\Lambda'}{\Rightarrow} \widetilde{q}$ such that $(\widetilde{p}, \widetilde{q}) \in \mathcal{R}$. Then, by Lemma A.3.1, $p' = (R)\widetilde{p}\sigma$, and there is Ω' that infers $(r)(p_1 \mid p_2) \stackrel{\Lambda}{\Rightarrow} (R)\widetilde{q}\sigma$ from $(r)(p_1 \mid p_2) \stackrel{\Lambda'}{\Rightarrow} \widetilde{q}$. The thesis follows from $\mathcal{R}$ being closed under renamings and addition of restrictions.

Now consider a transition $(r)(p_1 \mid p_2) \stackrel{\Lambda}{\Rightarrow} p'$. We prove the converse statement, again by cases on the last rule used to infer it:

Case (OPEN): Suppose it is inferred as follows

$$\dfrac{\dfrac{p_1 \stackrel{\Lambda_1}{\Rightarrow} p_1' \qquad p_2 \stackrel{\Lambda_2}{\Rightarrow} p_2'}{p_1 \mid p_2 \stackrel{\Lambda_1 \mid \Lambda_2}{\Longrightarrow} p_1' \mid p_2'}\,(\text{PAR})}{\begin{array}{c}\vdots\\ \Omega_1\\ \vdots\\ p_1 \mid p_2 \stackrel{\Lambda_1' \mid \Lambda_2}{\Longrightarrow} (R_1)(p_1' \mid p_2')\sigma_1\\ \vdots\\ \Omega_2\\ \vdots\end{array}}$$

$$\dfrac{p_1 \mid p_2 \stackrel{\Lambda'}{\Rightarrow} (R')(R_1)(p_1 \mid p_2)\sigma \circ \sigma_1}{(r)(p_1 \mid p_2) \stackrel{(r)(\Lambda' /\!\!/ r)}{\Longrightarrow} (R')(R_1)(p_1' \mid p_2')\sigma \circ \sigma_1}\,(\text{OPEN})$$

where Ω_1 contains instances of non-linear rules that only act on Λ_1 and Ω_2 all the other ones.

Now we want to move (OPEN) before (PAR), but we have to take care of the following situation: Ω_1 and Ω_2 may contain some occurrences of (COM) concatenating α_1 and α_2 such that $\alpha_1 \in \Lambda_1$, $\alpha_2 \in \Lambda_2$ and $r \in \mathrm{obj_{in}}(\alpha_2)$. Since we have to restrict r earlier in the proof, r will become bound in each α_1, so we must turn each α_2 into a bound input path via (iii) of Lemma A.3.2 in order for (COM) to be applied. Suppose Λ_2 contains n such input paths $a_i b_i(s_i); W_i; \bullet$. Let $s_1', \ldots, s_n'$ be fresh names, $\sigma_2 := [{}^{s_1'}/_{s_1}, \ldots, {}^{s_n'}/_{s_n}]$ and $\widehat{\sigma}_2 := [{}^{r_1}/_{s_1'}, \ldots, {}^{r_n}/_{s_n'}]$. By repeatedly applying (iii) of Lemma A.3.2 we get a transition $p_2\sigma_2^* \stackrel{\Lambda_2'}{\Rightarrow} p_2''$ such that $p_2''\widehat{\sigma}_2 = p_2'$, where Λ_2' is obtained by replacing each $a_i b_i r_i; W_i; \bullet \in \Lambda_2$ with $a_i b_i(s_i); W_i; \bullet$, for $i = 1, \ldots, n$. The new proof is

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{p_1 \overset{\Lambda_1}{\Longrightarrow} p_1'}{\vdots}\ \Omega_1'\ \vdots
}{p_1 \overset{\Lambda_1'}{\Longrightarrow} (R_1)p_1'\sigma_1}\ (\textsc{open})
}{(r)p_1 \overset{(r)(\Lambda_1' /\!\!/ r)}{\Longrightarrow} (R_1)p_1'\sigma_1 \qquad p_2\sigma_2^* \overset{\Lambda_2'}{\Longrightarrow} p_2''}\ (\textsc{par})
}{(r)p_1 \mid p_2 \overset{(r)(\Lambda_1' /\!\!/ r)\mid \Lambda_2'}{\Longrightarrow} (R_1)p_1'\sigma_1 \mid p_2''}
$$

$$
\vdots\ \Omega_2'\ \vdots
$$

$$
(r)p_1 \mid p_2\sigma_2^* \overset{(r)(\Lambda' /\!\!/ r)}{\Longrightarrow} (R')((R_1)p_1'\sigma_1 \mid p_2'')\sigma'
$$

where Ω_1' and Ω_2' contain instances of the same kind of rules, and concatenate the same pairs of paths, as Ω_1 and Ω_2, with the exception of input paths of the form $abr; W; \bullet$ in Ω_2', replaced by bound versions as described.

We now prove that old and new proof have matching processes in their roots: we have $(r)p_1 \mid p_2\sigma_2^* \equiv (r)p_1 \mid p_2$, by α-conversion, and

$$
\begin{aligned}
(R')((R_1)p_1'\sigma_1 \mid p_2'')\sigma' &= (R')((R_1)p_1'\sigma_1 \mid p_2'')\sigma \circ \widehat{\sigma}_2 \\
&= (R')((R_1)p_1'\sigma_1 \mid p_2''\sigma_2)\sigma && \text{by freshness of } \mathrm{dom}(\sigma_2) \text{ w.r.t. } p_1 \\
&= (R')((R_1)p_1'\sigma_1 \mid p_2')\sigma && \text{(iii) of Lemma A.3.2} \\
&= (R')((R_1)p_1' \mid p_2')\sigma \circ \sigma_1 && \text{by freshness of } \mathrm{dom}(\sigma_1) \text{ w.r.t. } p_2
\end{aligned}
$$

where the first equation comes by observing that σ', besides extrusions of the old proof, also handles ones for the additional bound input paths; the renaming of placeholders with extruded names for these new extrusions is exactly $\widehat{\sigma}_2$.

Finally, by closure under scope extension and renamings

$$
((R')(R_1)(p_1' \mid p_2')\sigma \circ \sigma_1, (R')((R_1)p_1' \mid p_2')\sigma \circ \sigma_1) \in \mathcal{R} \ .
$$

Case (RES): Analogous.

Non-linear rule: As before.

$\boxed{p \equiv p_1 \mid p_2, q \equiv p_2 \mid p_1}$ Suppose $(p_1 \mid p_2, p_2 \mid p_1) \in \mathcal{R}^2$ and $p_1 \mid p_2 \overset{\Lambda}{\Rightarrow} p'$. This transition can be inferred through (PAR) or a non-linear rule. In the first case, $p_2 \mid p_1$ is able to simulate it, by the commutativity of the parallel operator of concurrent paths. Its continuation is clearly paired with p' in $\mathcal{R}$. The second case is analogous to the "non-linear rule" cases shown above.

$\boxed{p \equiv (r)p', q \equiv (r)q'}$ Suppose $((r)p', (r)q') \in \mathcal{R}^3_{n+1}$ and $(r)p' \stackrel{\Lambda}{\Rightarrow} p''$, with $\mathrm{Bn}(\Lambda) \# (r)q'$. We have to prove that $(r)q'$ can simulate this transition. We proceed by cases on the last rule used to infer it.

Case (OPEN): then the transition is inferred from $p' \stackrel{\Lambda'}{\Rightarrow} p''$ and $\Lambda \equiv_\Lambda (r)(\Lambda' /\!\!/ r)$. Since $(p', q') \in \mathcal{R}_n$ and clearly $\mathrm{Bn}(\Lambda') \# q'$, by induction hypothesis $q' \stackrel{\Lambda'}{\Rightarrow} q''$, with $(p'', q'') \in \mathcal{R}$, so we can apply (OPEN) to get $(r)q' \stackrel{\Lambda}{\Rightarrow} q''$.

Case (RES): analogous to (OPEN); the main difference is that the two transitions have $(r)p''$ and $(r)q''$ as continuations, which are paired in $\mathcal{R}$ by its closure under addition of restrictions.

Case non-linear rule: Analogous to the "non-linear rule" cases above.

$\boxed{p \equiv p'\sigma, q \equiv q'\sigma}$ Suppose $(p'\sigma, q'\sigma) \in \mathcal{R}^4_{n+1}$ and $p'\sigma \stackrel{\Lambda}{\Rightarrow} p''$, with $\mathrm{Bn}(\Lambda) \# q'\sigma$. We can safely assume that

$$\mathrm{bn}(q'\sigma) \text{ are fresh w.r.t. } p'\sigma \text{ and viceversa} \tag{A.5}$$

(we can obtain this by α-conversion, which is contained in the bisimilarity). By Lemma A.3.5 and A.3.1 there is a transition $p' \stackrel{\widetilde{\Lambda}}{\Rightarrow} \widetilde{p}$ and a sequence Ω of non-linear rule instances such that, if we apply Ω to $p'\sigma \stackrel{\widetilde{\Lambda}\sigma}{\Longrightarrow} \widetilde{p}\sigma$, we get $p'\sigma \stackrel{\Lambda}{\Rightarrow} (R)(\widetilde{p}\sigma)\sigma'$, for some R and σ' according to Lemma A.3.1, where the continuation is p'' with some restrictions brought to the top level. In order to apply the induction hypothesis (i.e. that $\mathcal{R}_n$ is a bisimulation) to $p' \stackrel{\widetilde{\Lambda}}{\Rightarrow} \widetilde{p}$, we have to show that $\mathrm{Bn}(\widetilde{\Lambda}) \# q'$: this comes by observing that σ does not affect bound names (Convention A.1), so we have $\mathrm{bn}(p') \# q'$ by (A.5), which holds in particular for $\mathrm{Bn}(\widetilde{\Lambda})$ (as $\mathrm{Bn}(\widetilde{\Lambda}) \subseteq \mathrm{bn}(p')$).

Therefore $q' \stackrel{\widetilde{\Lambda}}{\Rightarrow} \widetilde{q}$, with $(\widetilde{p}, \widetilde{q}) \in \mathcal{R}$, which can be renamed to $q'\sigma \stackrel{\widetilde{\Lambda}\sigma}{\Longrightarrow} \widetilde{q}\sigma$, by (i) of Lemma A.3.2. By Lemma A.3.1 there are non-linear rules that act on this transitions by applying the same renaming and restricting the same names as Ω, yielding $q'\sigma \stackrel{\Lambda}{\Rightarrow} (R)(\widetilde{q}\sigma)\sigma'$. By closure under renamings and addition of restrictions we have $((R)(\widetilde{p}\sigma)\sigma', (R)(\widetilde{q}\sigma)\sigma') \in \mathcal{R}$ and, by closure under scope extension, we also have $(p'', (R)(\widetilde{p}\sigma)\sigma') \in \mathcal{R}$. Finally, by transitivity we can conclude $(p'', (R)(\widetilde{q}\sigma)\sigma') \in \mathcal{R}$. □

Proof of Theorem 5.2.7. By cases, considering all the operators: given an operator op, we prove that the relation

$$\mathcal{R} = \{(op(p), op(q)) \mid p \sim^{NC}_\kappa q\} \cup \sim^{NC}_\kappa$$

is a bisimulation.

Case op $\in \{\bar{\mathbf{a}}\mathbf{b}\mathbf{r}._-\,,\ \boldsymbol{\tau}._-\,,\ \mathbf{l_{ab}}._-\}$**:** given a pair of processes in $\mathcal{R}$, both can do the same paths, inferred through an axiom. Their continuations are just the unprefixed processes, which are bisimilar by definition of $\mathcal{R}$.

Case op $= \mathbf{a}(\mathbf{s})\_$**:** given $(a(s).p, a(s).q) \in \mathcal{R}$, both can do the same singleton free or bound input paths. The continuations are p and q, possibly renamed, which are again paired in $\mathcal{R}$, by Lemma A.3.6.

Case op $= \_ + \mathbf{q}$**:** consider $(p_1 + q, p_2 + q) \in \mathcal{R}$ and suppose $p_1 + q \overset{\Lambda}{\Rightarrow} p_1'$ is inferred using (SUM-L) from $p_1 \overset{\Lambda}{\Rightarrow} p_1'$. Then, by definition of $\mathcal{R}$, $p_2 \overset{\Lambda}{\Rightarrow} p_2'$, with $p_1' \sim_\kappa^{NC} p_2'$, so $p_2 + q \overset{\Lambda}{\Rightarrow} p_2'$, using (SUM-L), and $(p_1', p_2') \in \mathcal{R}$. The (SUM-R) case is obvious. If the transition is inferred via some non-linear rules, the usual argument applies: one finds the first occurrence of (SUM-L) above these non-linear rules, recovers a simulating transition of $p_2 + q$ as shown, and then uses Lemma A.3.1 on this transition to get a transition with label Λ. The non-linear rules add the same restrictions and renaming to p_1' and p_2', so the thesis follows by closure of $\sim_\kappa^{NC}$ under restrictions and renamings. The case $q + \_$ is analogous.

Case op $= \_ \mid \mathbf{q}$**:** here we prove the claim for the relation

$$\mathcal{R}' := \bigcup_q \{(p_1 \mid q, p_2 \mid q) \mid p_1 \sim_\kappa^{NC} p_2\} \cup \sim_\kappa^{NC} \quad .$$

Consider $(p_1 \mid q, p_2 \mid q) \in \mathcal{R}'$ and suppose $p_1 \mid q \overset{\Lambda_1 \mid \Lambda_2}{\Longrightarrow} p_1' \mid q'$, with $\mathrm{Bn}(\Lambda_1 \mid \Lambda_2) \# p_2 \mid q$, is inferred from $p_1 \overset{\Lambda_1}{\Rightarrow} p_1'$ and $q \overset{\Lambda_2}{\Rightarrow} q'$ using (PAR). Since $p_1 \sim_\kappa^{NC} p_2$ and $\mathrm{Bn}(\Lambda_1) \# p_2$, we also have $p_2 \overset{\Lambda_1}{\Rightarrow} p_2'$, so we can apply (PAR) and get $p_2 \mid q \overset{\Lambda_1 \mid \Lambda_2}{\Longrightarrow} p_2' \mid q'$. Finally, from $p_2' \sim_\kappa^{NC} p_1'$ it follows that $(p_1' \mid q', p_2' \mid q') \in \mathcal{R}'$. If the transition is inferred through non-linear rules, the usual argument applies. The case $q \mid \_$ is analogous.

Case op $= (\mathbf{r})\_$**:** directly by Lemma A.3.6.

□

A.4 Proofs for Chapter 6

A.4.1 Proof of Theorem 6.3.1

By induction on the number of peers. If there are no peers, the claim is trivially true, because a_0 is the only responsible for every k.

If there are n peers, suppose a_{n+1} triggers the join procedure by sending $\mathbf{join\_req}[a_{n+1}]$ to a_{i_0}. Let us consider the most general case: when $shl(a_{i_0}, a_{n+1}) = 0$ and a_{n+1} does not belong to the interval spanned by a_{i_0}'s leaf-set. Then, when a_{i_0} receives the join request, the last **else** branch in the **join_req** handler (see Figure 6.2) is taken and the identifiers in a_{i_0}'s routing table are sent back to a. Now, since $shl(a, a_{i_0}) = 0$, only the first row of a_{n+1}'s routing-table, namely the one formed by cells with identifiers $c_{d_1}, \ldots, c_{d_\sigma}$, is filled. Then the join request is forwarded to a sequence of peers $a_{i_1}, a_{i_2}, \ldots$, and each acknowledgement allows a_{n+1} to fill a single row of its routing table. In fact, by induction hypothesis, $a_{i_{j+1}}$ shares one

more digit than a_{i_j} with a_{n+1}, so it will provide identifiers for the $j+2$-th row of a's routing table. Eventually the closest peer to a_{n+1}, say c, is reached. This will let a populate its leaf-set and possibly the last row of its routing table. Now we have to show that each a_i is able to forward messages for any key k to some b satisfying Property 6.1.2. We distinguish two cases:

$i = n+1$: Suppose k is in the range of a_{n+1}'s leaf-set, then the forwarding happens along $a \boxminus b$ in $\mathcal{L}_{LS}^{a_{n+1}}$, but b is also target of a link in $\mathcal{L}_{LS}^{c}$, so the induction hypothesis applies. Otherwise, if $shl(k, a_{n+1}) = j$, then $a \boxdot b$ belong to the $j+1$-th row of a_{n+1}'s routing table, but b is also target of a link in $\mathcal{L}_{RT}^{a_{i_j}}$; again, the claim follows by induction hypothesis.

$1 \leq i \leq n$: if a_{n+1} is not responsible for k, then the induction hypothesis gives the claim. Otherwise, suppose a_{n+1} is the new responsible for k and let r_k be the old one. The claim is clearly true for $a_i \neq r_k, i = 1, \ldots, n$, because it already has a link that gets closer to k. Instead, we have to verify the claim for r_k, because it did not have any such link before a_{n+1} joined the ring. The key observation is that r_k is in a_{n+1}'s leaf-set, so it receives a **join_ntf**$[a_{n+1}]$ message, sent by a_{n+1} at the end of its join procedure. This message triggers the creation of a new link to a_{n+1}, which becomes part of $\mathcal{L}_{LS}^{r_k}$ and/or $\mathcal{L}_{LS}^{r_k}$. This link satisfies the theorem's requirements, because a_{n+1} is closer to k than r_k.

A.4.2 Proof of Lemma 6.4.1

If a it the responsible for k, then $\texttt{Entry}(k, v, a)$ gives the required transition. Otherwise, this is a transition of `Route` within $\texttt{Peer}(a)$. In fact, thanks to Theorem 6.3.1, this process is able to activate a link that satisfies Property 6.1.2.

Bibliography

[1] Openflow foundation website. http://www.openflow.org/. 4

[2] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001. 2

[3] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999. 2

[4] Jiří Adámek. Introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005. 2, 23

[5] Jirí Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A coalgebraic perspective on minimization and determinization. In *FoSSaCS*, pages 58–73, 2012. 2

[6] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994. 21, 22

[7] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011. 2, 6, 98

[8] Filippo Bonchi, Maria Grazia Buscemi, Vincenzo Ciancia, and Fabio Gadducci. A presheaf environment for the explicit fusion calculus. *J. Autom. Reasoning*, 49(2):161–183, 2012. 9, 50, 63, 96

[9] Filippo Bonchi, Barbara König, and Ugo Montanari. Saturated semantics for reactive systems. In *LICS*, pages 69–80, 2006. 63

[10] Filippo Bonchi and Ugo Montanari. Reactive systems, (semi-)saturated semantics and coalgebras on presheaves. *Theor. Comput. Sci.*, 410(41):4044–4066, 2009. iv, 7, 63

[11] Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Inf.*, 35(5):353–400, 1998. 97

[12] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *ESOP*, pages 18–32, 2007. 2

[13] Nadia Busi and Roberto Gorrieri. Distributed semantics for the π-calculus based on petri nets with inhibitor arcs. *J. Log. Algebr. Program.*, 78(3):138–162, 2009. 96

[14] Andrew T. Campbell and Irene Katzela. Open signaling for atm, internet and mobile networks (opensig'98), 1999. 4

[15] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Electr. Notes Theor. Comput. Sci.*, 68(2):15–32, 2002. 6

[16] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000. 2

[17] Vincenzo Ciancia, Alexander Kurz, and Ugo Montanari. Families of symmetries as efficient models of resource binding. *Electr. Notes Theor. Comput. Sci.*, 264(2):63–81, 2010. iv, 4, 50, 51, 60, 61

[18] Vincenzo Ciancia and Ugo Montanari. Symmetries, local names and dynamic (de)-allocation of names. *Inf. Comput.*, 208(12):1349 – 1367, 2010. 59

[19] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. Klaim: A kernel language for agents interaction and mobility. *IEEE Trans. Software Eng.*, 24(5):315–330, 1998. 95

[20] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. On the expressive power of klaim-based calculi. *Theor. Comput. Sci.*, 356(3):387–421, 2006. 95

[21] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calculus for global computing. *Inf. Comput.*, 205(10):1491 – 1525, 2007. 5, 6, 9, 10, 95

[22] Edsko De Vries, Adrian Francalanza, and Matthew Hennessy. Reasoning about explicit resource management (extended abstract). In *PLACES*, pages 15–21. ETAPS, April 2011. 96

[23] Pierpaolo Degano and Corrado Priami. Causality for mobile processes. In Zoltán Fülöp and Ferenc Gécseg, editors, *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 660–671. Springer, 1995. 97

[24] Gian Luigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In *FMCO*, pages 22–43, 2005. 96

[25] Gian Luigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimization of hd-automata for the π-calculus using polymorphic types. *Theor. Comput. Sci.*, 331(2-3):325–365, 2005. 4

[26] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *LICS*, pages 193–202, 1999. 29

[27] Marcelo P. Fiore and Sam Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, 2006. 28

[28] Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *LICS*, pages 93–104, 2001. iv, 3, 4, 7, 9, 11, 26, 29, 30, 32, 33, 49, 50, 58, 63, 96, 97

[29] Adrian Francalanza and Matthew Hennessy. A theory of system behaviour in the presence of node and link failure. *Inf. Comput.*, 206(6):711 – 759, 2008. 5, 6, 9, 10, 95

[30] Fabio Gadducci, Marino Miculan, and Ugo Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher-Order and Symbolic Computation*, 19(2-3):283–304, 2006. 29, 51, 60

[31] Neil Ghani, Kidane Yemane, and Björn Victor. Relationally staged computations in calculi of mobile processes. *Electr. Notes Theor. Comput. Sci.*, 106:105–120, 2004. 96

[32] Matthew Hennessy. A calculus for costed computations. *Logical Methods in Computer Science*, 7(1), 2011. 96

[33] Matthew Hennessy and Manish Gaur. Counting the cost in the π-calculus. *Electr. Notes Theor. Comput. Sci.*, 229(3):117–129, 2009. 96

[34] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985. 2

[35] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002. 95

[36] Dan Hirsch and Ugo Montanari. Synchronized hyperedge replacement with name mobility. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *CONCUR*, volume 2154 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 2001. 96

[37] Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. 1

[38] Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford Logic Guides. Clarendon Press, 2002. 20, 28, 54

[39] Paris C. Kanellakis and Scott A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990. 2

[40] Bartek Klin. Coalgebraic modal logic beyond sets. *Electr. Notes Theor. Comput. Sci.*, 173:177–201, 2007. 2

[41] Saunders M. Lane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1998. 22, 23, 28, 50, 108

[42] Ivan Lanese. *Synchronization strategies for global computing models*. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy, 2006. 10

[43] Ivan Lanese. Concurrent and located synchronizations in π-calculus. In *SOFSEM (1)*, pages 388–399, 2007. 10, 77, 96

[44] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory (Universitext)*. Springer, 1992. 101

[45] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker, and Jonathan S. Turner. Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74, 2008. 4

[46] Marino Miculan. A categorical model of the fusion calculus. *Electr. Notes Theor. Comput. Sci.*, 218:275–293, 2008. 96

[47] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. 1, 4

[48] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i/ii. *Inf. Comput.*, 100(1):1–77, 1992. 1, 4, 13, 99

[49] Ugo Montanari and Marco Pistore. Concurrent semantics for the π-calculus. *Electr. Notes Theor. Comput. Sci.*, 1:411–429, 1995. 96

[50] Ugo Montanari and Marco Pistore. Structured coalgebras and minimal hd-automata for the π-calculus. *Theor. Comput. Sci.*, 340(3):539–576, 2005. iv, 4

[51] Ugo Montanari and Matteo Sammartino. Network conscious π-calculus. Technical Report TR-12-01, Computer Science Department, University of Pisa, February 2012. 8, 11

[52] Ugo Montanari and Matteo Sammartino. Network conscious π-calculus: A concurrent semantics. *Electr. Notes Theor. Comput. Sci.*, 286:291–306, 2012. 8, 10

[53] Ugo Montanari and Matteo Sammartino. A network-conscious π-calculus and its coalgebraic semantics. *To appear in Theor. Comput. Sci.*, 2013. 8, 9, 11

[54] Ugo Montanari and Vladimiro Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *Fundam. Inform.*, 16(1):171–199, 1992. 9

[55] Lawrence S. Moss. Coalgebraic logic. *Ann. Pure Appl. Logic*, 96(1-3):277–317, 1999. 2

[56] Frank J. Oles. *Type algebras, functor categories and block structure*. Cambridge University Press, 1986. 4

[57] Joachim Parrow. *An Introduction to the π-calculus*, pages 479–543. 2001. 13

[58] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *LICS*, pages 176–185, 1998. 2

[59] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001. iii, 8, 81, 82

[60] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. 2, 23

[61] Davide Sangiorgi. Locality and true-concurrency in calculi for mobile processes. In Masami Hagiya and John C. Mitchell, editors, *TACS*, volume 789 of *Lecture Notes in Computer Science*, pages 405–424. Springer, 1994. 97

[62] Davide Sangiorgi and David Walker. *π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001. 13

[63] Sam Staton. Name-passing process calculi: operational models and structural operational semantics. Technical Report 688, University of Cambridge, 2007. 20, 26, 28, 32, 50, 51, 58

[64] Sam Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, 7(1), 2011. 2

[65] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26:5–18, 1996. 4

[66] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291, 1997. 10

[67] Y.Rekhter. A border gateway protocol 4 (bgp-4). http://www.ietf.org/rfc/rfc1771.txt, March 1995. 67, 78