Unique solution techniques for processes and functions

Adrien Durier



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Numéro National de Thèse : 2020LYSEN016

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée par
l'Ecole Normale Supérieure de Lyon
en cotutelle avec
Università di Bologna

Ecole Doctorale N°512
École Doctorale en Informatique et Mathématiques de Lyon

Discipline : Informatique

Soutenue publiquement le 11/06/2020, par :
Adrien Émile DURIER

Unique solution techniques for processes and functions

Techniques d'unicité des solutions pour processus et fonctions

---

Devant le jury composé de :

Kesner, Delia	IRIF	Rapporteure
Roscoe, Bill	University of Oxford	Rapporteur
Dezani, Mariangiola	Università di Torino	Examinatrice
Schmitt, Alan	INRIA – IRISA	Examinateur
Hirschkoff, Daniel	ENS de Lyon	Directeur de thèse
Sangiorgi, Davide	Università di Bologna	Co-tuteur de thèse

Abstracts

Abstract

The bisimulation proof method is a landmark of the theory of concurrency and programming languages: it is a proof technique used to establish that two programs, or two distributed protocols, are equal, meaning that they can be freely substituted for one another without modifying the global observable behaviour. Such proofs are often difficult and tedious; hence, many proof techniques have been proposed to enhance this method, simplifying said proofs. We study such a technique based on 'unique solution of equations'. In order to prove that two programs are equal, we show that they are solution of the same recursive equation, as long as the equation has the 'unique solution property': two of its solutions are always equal. We propose a guarantee to ensure that such equations do have a unique solution. We test this technique against a long-standing open problem: the problem of full abstraction for Milner's encoding of the call-by-value λ-calculus in the π-calculus.

Résumé

La méthode de preuve par bisimulation est un pilier de la théorie de la concurrence et des langages de programmation. Cette technique permet d'établir que deux programmes, ou deux protocoles distribués, sont égaux, au sens où l'on peut substituer l'un par l'autre sans affecter le comportement global du système. Les preuves par bisimulation sont souvent difficiles et techniquement complexes. De ce fait, diverses techniques ont été proposées pour faciliter de telles preuve. Dans cette thèse, nous étudions une telle technique de preuve pour la bisimulation, fondée sur l'unicité des solutions d'équations. Pour démontrer que deux programmes sont égaux, on prouve qu'ils sont solution de la même équation, à condition que l'équation satisfasse la propriété d'unicité des solutions : deux solutions de l'équation sont nécessairement égales. Nous utilisons cette technique pour répondre à une question ouverte, à savoir le problème de *full abstraction* pour l'encodage, dû à Milner, du λ-calcul en appel par valeur dans le π-calcul.

Riassunto

La bisimulazione è una tecnica di prova fondamentale in teoria della concorrenza e dei linguaggi di programmazione. Questa tecnica viene usata per dimostrare che due programmi, o due protocolli distribuiti, sono uguali, nel senso che l'uno può sostituire l'altro senza modificare il comportamento globale del sistema. Le prove di bisimulazione sono spesso difficili e tecnicamente pesanti. Per questa ragione, varie tecniche di prova sono state introdotte per facilitare le prove di bisimulazione. In questo documento viene studiata tale tecnica, che sfrutta l'unicità delle soluzioni di equazioni. Per dimostrare che due programmi sono uguali, si stabilisce che sono soluzioni della stessa equazione ricorsiva, dal momento in cui l'equazione soddisfa una proprietà di "unicità delle soluzioni": ogni due soluzioni di questa equazione sono uguali. Questa tecnica viene usata per rispondere alla questione della *full abstraction* per l'encodaggio del λ-calcolo in call-by-value nel π-calcolo, proposto inizialmente da R. Milner.

Introduction

Guardedness & 'Unique Solution of equations'

Guardedness and coinduction. The general subject of this thesis is the study of programming languages as formal objects, and in particular concurrent process algebras – stripped-down formalisms used to represent concurrent and interactive processes. In order to define such formal objects, we need well-formedness guarantees. The kind of mathematical abstractions that we consider include not only programs written in a formal programming language, or concurrent processes in a process algebra, but logical proofs as well. Among the most standard tools of the mathematician for defining these objects are induction (for bottom-up, finite, constructions), and its dual, coinduction (for top-down, infinite, constructions).

In the case of coinduction, we often work with infinite objects. It is thus of critical importance to enforce that the definitions do make sense by not containing trivial loops. For instance, we do not want that the conclusion of a proof is derived directly from that same assumption itself – that we assumed to hold as a co-inductive hypothesis. Given some logical assertion H, the following (infinite) proof should not be valid:

$$\dfrac{\dfrac{\dfrac{\vdots}{H}}{H}}{H}$$

as otherwise any assertion could be proven in such a fashion.

Guardedness is a tool to study (co-)recursive definitions, the existence of such defined objects, and their uniqueness. The idea of guardedness is to use precedence to enforce the *productivity* of definitions: before the recursion or co-recursion takes place, something has to happen; for instance, another logical rule has to be used, or the program has to perform an operation (it produces something). When defining an inductive or a coinductive object (a fixed point), for instance, a proof, or a program's behaviour, we say that it is guarded whenever it enforces the precedence of a productive operation over the (co-)recursion (often through the means of a syntactic construct). Guardedness has been used to study and warrant co-inductive logical proofs, for the modal μ-calculus [DHL06] or in proof assistants such as coq [Coq93, Gim94], and more recently for parameterized coinductive proofs [HNDV13]. It also has been an integral part of the theory of programming languages and of concurrency since their very beginning [BBR10, Hoa85, Mil89].

Guardedness in process algebras. We illustrate the concept of guardedness within the framework of process algebras. Consider some processes that we write $P, Q, \ldots$ These processes may perform some *actions*. We represent these actions with letters $a, b, c \ldots$ We write $P \xrightarrow{a} P'$ when the process P performs the action a, and reduces to the process P', which may then perform additional actions. In that simple setting, the concept of guardedness is captured by a simple syntactic operator, the prefix operator. We write $a.\,P$ for the process P prefixed by the action a: the process $a.\,P$ has to first perform the action a, leaving then the residual process P. With the previous notation, this is implemented by the rule :

$$a.\,P \xrightarrow{a} P$$

We may now define a co-inductive process, using only this syntax. The following recursive definition

$$P \stackrel{\triangle}{=} a.\,P$$

is, indeed, well-formed: there is a unique process that satisfies this definition. Such a process may only perform the action a, and it may perform it indefinitely, over and over again: $P \xrightarrow{a} P \xrightarrow{a} P \xrightarrow{a} \ldots$ In contrast, the definition $P \stackrel{\triangle}{=} P$ is not well-formed: it is unclear which actions such a P should be able to perform, if any. And, indeed, any process satisfies this equation. Hence, the problem of well-formedness for such defined processes boils down to the property of 'unique solution' for their defining equation, i.e., whether two or more different processes may be solution of said equation – as long as at least one such solution exists. Writing X for a process variable (a place-holder for a process), the equation $X = X$ indeed does not enjoy the unique-solution property, while the equation $X = a.\,X$ does.

The proof technique of unique solution of equations. We study guardedness because it provides a tool to enforce that equations have a unique solution. However, we are mostly interested in unique solution of equations as a *proof technique*, rather than a tool to provide proper definitions. Proof techniques are an essential ingredient of the theory of any formal language, including process algebras. Consider the following toy language, for a given set of names $\mathcal{N}$:

$$P, Q \;:=\; a.\,P \mid \mathbf{0} \quad (a \in \mathcal{N})$$

The null process $\mathbf{0}$ is a process that may not perform any action. We use the names in the set $\mathcal{N}$ to represent different actions than processes may perform. As above, we use a Labeled Transition System (LTS) to describe the behaviour of processes: for each name $a, b, \cdots \in \mathcal{N}$ we define a relation $\xrightarrow{a}$ over processes. Using infix notation for these relations, we write $P \xrightarrow{a} P'$ when P can perform the action a; leaving P', in turn, able to perform some actions. By definition, the null process has no possible actions: for any a and Q, $\mathbf{0} \not\xrightarrow{a} Q$. This is illustrated by the following process behaviour:

$$a.\,b.\,\mathbf{0} \xrightarrow{a} b.\,\mathbf{0} \xrightarrow{b} \mathbf{0}$$

Now that we gave a semantics to the language, describing the behaviour of processes, we want to equate processes that exhibit the same behaviour, even if they do not share the

same syntax. Consider, for instance, recursively defined processes $P \triangleq a.\, P$ and $Q \triangleq a.\, a.\, Q$ (leaving aside, for now, the formal meaning of such recursive definitions). Both have the same behaviour: indeed, they might only perform the action a, infinitely many times. We thus want to show that $P = Q$. By equality we do not mean syntactic equality: consider that $=$ stands for some equivalence relation that captures the behaviour of processes (we call such relations 'behavioural equivalences'). To provide a proof that they do indeed have the same behaviour, we can use the fact that the equation

$$X = a.\, a.\, X$$

has a unique solution: if P and Q are solutions of this equation, then we can deduce that $P = Q$. Indeed, as a consequence of guardedness, this equation has a unique solution. It is only left to show that $P = a.\, a.\, P$ and $Q = a.\, a.\, Q$. The second equality is a direct consequence of Q's definition. On the other hand, it is not hard to show that $P = a.\, a.\, P$: both P and $a.\, a.\, P$ can perform two transitions a, to reduce to P. The following diagram illustrates this proof:

$$\begin{array}{ccc} P & = & a.\, a.\, P \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle a} \\ P & = & a.\, P \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle a} \\ P & = & P \end{array}$$

We call this procedure for proving an equality the 'unique solution of equations' technique. It has been proposed by Milner in the setting of CCS, and plays a prominent role in proofs of examples in his book [Mil89]. It was independently proposed by Hoare for his language CSP [Hoa85]. The method is important in verification techniques and tools based on algebraic reasoning [Ros10, BBR10, GM14].

Guardedness in presence of parallel composition. Process algebras are mostly used to represent concurrent executions and communication. We thus assume action $P \xrightarrow{a} P'$ stands for the reception of a message, on some communication channel a. This is an external action: we assume that in presence of another process, emitting some message (we write $Q \xrightarrow{\overline{a}} Q'$ for the output on a), the two processes will synchronise, turning both external actions into a single internal action – invisible to the outside world. Thus, the guard of the process $a.\, P$, in presence of a message output on a, might be erased, leaving an unguarded process.

We introduce CCS, a very simple language used to represent basic concurrent processes and communications. It was first proposed by Milner [Mil89]. In CCS, processes may only perform two types of actions: receive a message on some channel (channels are represented by names, $a, b, \dots$), or output a message on such a channel. We write $a.\, P$ for the input prefix action, and $\overline{a}.\, P$ for the output prefix action. The null process is still $\mathbf{0}$. The new construct is parallel composition $P \mid Q$, that allows concurrent execution and synchronisation. This forms a subset of the original CCS; for now, we only consider this restricted syntax:

$$P, Q \;:=\; \mathbf{0} \;\mid\; a.\,P \;\mid\; \overline{a}.\,P \;\mid\; P \mid Q$$

In the process $a.\,P \mid b.\,Q$, the inputs a and b may occur in any order, and likewise, a may occur before or after any action of Q – as long as a occurs before any action of P, and b before any action of Q (prefixes still enforce precedence). Furthermore, input actions a and output actions $\overline{a}$ on the same channels *may* now synchronise, when put in parallel: in $a.\,\mathbf{0} \mid \overline{a}.\,\mathbf{0}$, an (internal) communication may occur, leaving the process $\mathbf{0} \mid \mathbf{0}$.

In presence of parallel composition, guardedness does not ensure uniqueness of solution anymore, as illustrated by the following equation:

$$X = a.\,X \mid \overline{a}.\,\mathbf{0} \tag{1}$$

Indeed, while this equation is syntactically guarded (the equation variable X is placed below a prefix operator), it has multiple solutions. The idea is that the $a.$ prefix does not enforce the behaviour of the equation's solution, as it may be removed through synchronisation. To take this into account, we distinguish between *strong equivalences* and *weak equivalences*.

Strong and weak equivalences. To model communications, we extend the Labeled Transition System with a new label, τ, standing for internal actions; thus $P \xrightarrow{\tau} P'$ means that P performs some internal computation, then reduces to P'. Likewise, we introduce the τ prefix: $\tau.\,P$ stands for some internal computation followed by the process P Such internal computation should not be observable by the outside world, and thus our behavioural equivalence should not account for τ-transitions. Synchronisations are described by the following τ-transition:

$$a.\,P \mid \overline{a}.\,Q \xrightarrow{\tau} P \mid Q$$

Note that such a transition is optional, as the left-hand process might still perform either the input or the output on a; for instance: $a.\,P \mid \overline{a}.\,Q \xrightarrow{a} P \mid \overline{a}.\,Q$.

We call a behavioural equivalence *strong* when it accounts for τ transitions, and *weak* when it does not. In this thesis, we are mostly interested in weak equivalences, as they are, in practice, the most relevant ones: e.g., two equivalent programs may produce the same result with different numbers of evaluation steps. However, the theory of a strong equivalence is usually more accessible, as illustrated by the simple guard condition for strong equivalences, sufficient to ensure that an equation has a unique solution. We are therefore interested in comparing the theories of unique solution of equations for strong and weak equivalences.

An instance of such equivalences are the trace equivalences. In the 'strong' trace equivalence, two processes are equated when they can perform the same sequences of actions (including τs). The weak variant of trace equivalence matches sequences of transitions, but ignoring τ-transitions. We now write a for the process $a.\,\mathbf{0}$ (likewise for τ and $\overline{a}$). Consider the two processes $\tau \mid \tau.\,a.\,a$ and $\tau.\,a \mid \tau.\,a$. For the strong trace equivalence, these processes are not equal: the first process produces the trace $\tau a a \tau$, while the second may not. However,

they are equal for weak trace equivalence, as their only weak traces are a and aa (τs are ignored).

Milner shows in his book [Mil89] on CCS that, when considering strong equivalences, unique solution of equations holds for any guarded equation. More precisely, he shows that this is the case for strong bisimilarity, an equivalence finer than trace equivalence; however, his result also holds for strong trace equivalence, and most strong behavioural equivalences. Indeed, the equation

$$X = \tau.X$$

has a unique solution for a strong equivalence (the solution is a process whose traces are finite sequences of τs). On the other hand, any process is solution for a weak equivalence, making this equation akin to $X = X$. This is illustrated by Equation (1) above: for any process P, we have the transition $a.P \mid \overline{a}.\mathbf{0} \xrightarrow{\tau} P \mid \mathbf{0}$. Thus, the behaviour of any solution will be expressed in the behaviour of $a.P \mid \overline{a}.\mathbf{0}$: while any solution must still have certain behaviours imposed by the equation, it may also have, in addition, any other behaviour (it may perform any other actions). In other words, if P is solution of this equation, so is $P \mid Q$ for any Q.

Because the transition $a.P \mid \overline{a}.\mathbf{0} \xrightarrow{\tau} P \mid \mathbf{0}$ holds for all possible solutions to the equation, what we did amounts to rewriting the body of the equation: somehow, $a.X \mid \overline{a}.\mathbf{0} \xrightarrow{\tau} X \mid \mathbf{0}$. Any process is solution of $X = X \mid \mathbf{0}$, and from this we deduce that solutions of the equation might exhibit any transition (but not that any process is solution, as this τ-transition is not the only possible transition of the original equation. In other words, the prefix guarding the equation can always be deleted, thus the equation fails to have a unique solution. However, for a strong equivalence, Equation (1) still has a unique solution.

Unique solution of equations for weak equivalences. Several notions of guardedness, strong enough to ensure uniqueness of solutions, have been studied, for different languages and process algebras; see, e.g., [BW90, GM14]. As shown above, the syntactic notion of guardedness from CCS is not sufficient. Hence, in Milner's theorem [Mil89], uniqueness of solutions relies on an additional limitation: the equations must be 'sequential', that is, the variables of the equations may not be preceded, in the syntax tree, by the parallel composition operator. This limits the expressiveness of the technique (in general, occurrences of the parallel composition operator cannot be removed), and its transport onto other languages (e.g., many languages, such as our restricted version of CCS, languages for distributed systems or higher-order languages, do not include any non-deterministic operator other than the parallel composition, which makes the theorem essentially useless). A comparable technique, involving similar limitations, has been proposed by Hoare in his book about CSP [Hoa85], and plays an equally essential role in the theory of CSP.

Contractions. In order to overcome such limitations, a variant of the technique, called *unique solution of contractions*, has been proposed [San15]. The technique is for behavioural equivalences; however the meaning of 'solution' is defined in terms of the contraction of the chosen equivalence. Contraction is, intuitively, a preorder that conveys an idea of efficiency

on processes, where efficiency is measured on the number of internal actions needed to perform a certain activity. The condition for applicability of the technique is, as for Milner's, purely syntactic: each variable in the body of an equation should be underneath a prefix. The technique has two main disadvantages:

1. the equational theory of the contraction preorder associated to an equivalence is not the same as the equational theory of the equivalence itself, which thus needs to be studied as well;

2. the contraction preorder is strictly finer than the equivalence, hence there are equivalent processes, one of which might be solution of a given contraction, while the other is not, and the technique might not be applicable in this case.

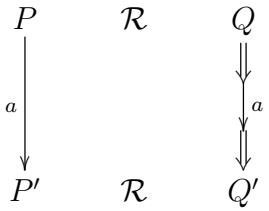
Roscoe's unique-solution technique. Roscoe [Ros97, Ros92], proposes a different approach for CSP, relying on the concept of *divergence*: a process diverges if it may produce, at any point of its execution, an infinite sequence of τ-transitions. For instance, the solution for strong equivalences of the equation $X = \tau.X$ produces a divergence. The unique-solution theorem condition relies on non-divergence of the *syntactic solution*, e.g., the solution of the equation defined recursively thanks to the equation: for instance the process $P \stackrel{\triangle}{=} \tau.P$ for the equation $X = \tau.X$. A similar approach is to consider the *infinite unfolding* of the equation: the infinite syntactic object obtained by replacing the variables with the body of the equation within the equation itself, infinitely many times (the infinite unfolding of the equation $X = \tau.X$ would be $\tau.\tau.\tau\ldots$). The unique-solution theorem essentially states that a guarded equation (or system of equations) whose infinite unfolding never produces a divergence has the unique-solution property. Roscoe's result is presented, as usual in CSP, with respect to denotational semantics and failure based equivalence [BHR84, BR84]. In such a setting, where divergence is catastrophic (e.g., it is the bottom element of the domain), the theorem has an elegant and natural formulation. (Indeed, Roscoe develops a denotational model [Ros92] in which the proof of the theorem fits within just a few lines.) Operational approaches, while more involved, are often more flexible, as they scale more easily to different languages and equivalences.

Bisimulation and up-to-context techniques

Bisimulation and bisimilarity. Bisimilarity is a behavioural equivalence, but also a mean to reason about behavioural equivalence. It originates in Milner's work on concurrency theory, but is now widely used in formal methods for programming languages – the certified compiler CompCert [Ler], for instance, relies heavily on simulation methods.

The popularity of bisimilarity is mostly due to its associated coinductive proof method – the bisimulation proof method. Two processes are deemed bisimilar if a bisimulation relation $\mathcal{R}$ relating them can be exhibited, where a bisimulation is a relation required to be closed under the following game: if $P\ \mathcal{R}\ Q$ (meaning P and Q are related by the bisimulation $\mathcal{R}$),

then for any action, say a, that P can perform, reducing to a residual process P' ($P \xrightarrow{a} P'$), then Q must be able to perform the same action a, reducing to a residual process Q', whereby P' and Q' are still related by $\mathcal{R}$. Because we consider weak equivalences, Q might perform any amount of internal work before and after performing a. Likewise, for any action emanating from Q: P must answer with the same action, and possibly some internal steps before and after. This can be described by the following diagram, where $\mathcal{R}$ is a a bisimulation relation, and $\Rightarrow$ is used to represent internal work (sequences of τ transitions) performed by the process.

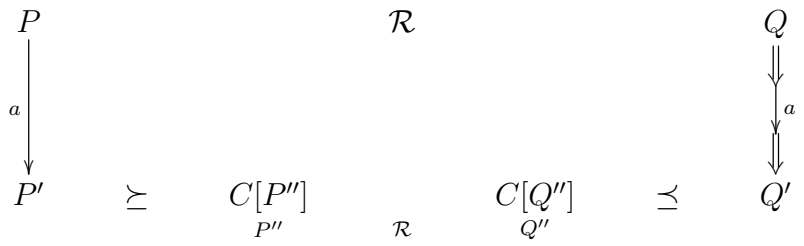


(for the actions of Q, the symmetrical game is played). When the challenge is a τ-transition, the answering process might perform any number of τ-steps, including zero.

Up-to techniques. Improvements to the bisimulation proof method are an active research area, and the best known such enhancements are the so-called '*up-to techniques*' [PS11]. Using such techniques, the derivatives of processes related by a bisimulation (P' and Q' in the diagram above) can be manipulated and rewritten, before they are required to be related by the bisimulation relation. The goal is to work with smaller relations, contained within bisimulation relations. Among such techniques, a most notable one is the 'up-to bisimilarity' technique, which allows to rewrite the residual processes using already-proven bisimilarity equalities: we only require that P' and Q' are bisimilar to processes related by $\mathcal{R}$. Unfortunately, this technique has been proven to be unsound [SM92]. Indeed, using weak bisimilarity, it is possible to add new internal transitions after every step, so that the bisimulation game never reaches the point at which the considered processes differ.

We therefore have to rely on weaker techniques, such as the so-called 'up-to expansion' techniques, which restricts the up-to bisimilarity technique (expansion is a preorder that refines bisimilarity). Find new and more powerful replacements for the up-to bisimilarity technique has been an active research area [San15, Pou08].

Up-to-context techniques. Among the most powerful such forms of enhancement, another is the up-to-context technique, that allows one to exploit the syntactic structures of the language by removing a common context in the derivative terms, and requiring only the resulting terms to be related (the context may have multiple holes, in which case the tuples of resulting terms should be componentwise related). It is rarely the case that the derivative terms explicitly exhibit a common context; usually they have to be 'massaged' (i.e., by applying some algebraic laws) in order to bring up such a common context. A common way of achieving this is by combining up-to context with the above up-to-expansion technique. Writing $\succeq$ for the expansion, and $C[P'']$ for the process P in some context C (shared with the process $C[Q'']$), the bisimulation game becomes:



As expressed by the diagram above, we can replace, modulo expansion, the process P' resulting from the transition with $C[P'']$, and similarly Q' with $C[Q'']$. We can then remove the common context C, and compare P'' and Q'' using $\mathcal{R}$.

Contractions and up to context. In his work on unique solution of contractions [San15], Sangiorgi illustrates an interesting correspondence between the unique-solution and the up-to-context techniques. Indeed, consider a proof of equivalence of the tuples of processes $\widetilde{P}$ and $\widetilde{Q}$, using the system of equations $\widetilde{X} = \widetilde{E}$, assuming it has a unique solution. To show that both $\widetilde{P}$ and $\widetilde{Q}$ are solutions of the system of equations, we have to show that for each pair $P_i \ \mathcal{R} \ Q_i$, P_i and Q_i can be rewritten to be processes equipped with some common context, given by the body of the corresponding equation E_i. This is precisely the idea behind up-to-context techniques. In other words, the body of an equation acts like a context that is erased in a proof using 'up to context'.

This correspondence particularly holds true in CCS (and other first-order languages): Sangiorgi shows that, for any equivalence proof made with a system of equation-contractions, there is a proof of similar complexity using a bisimulation up to context; this means the bisimulation-up-to relation is of the same size as the system of equations. The converse also holds: bisimulations up to can be turned into systems of equation-contractions. However, this perfect correspondence does not hold anymore when considering higher-order languages, or languages with name-passing, such as the π-calculus.

Name-passing and Higher-Order

The π-calculus. The π-calculus is an extension of CCS, where channel names can be exchanged along communications. The other operators of the language are the same as in CCS. However, despite this, the theory of the language is substantially impacted by this new ability. In many regards, the theory of the π-calculus is closer to the theory of the λ-calculus: many problems arise that are simply absent from CCS.

The names that are exchanged are the very same names used as prefixes. $a(b).\,P$ is the prefix representing the reception of a name b during a communication whose subject is a – thus, b is bound in $a(b).\,P$. Consider the process $a(b).\,b$. It receives a name b along a communication on a, then listens on this newly received channel (not that we are using here CCS-like syntax, and write simply b, using the fact that the information sent on channel b is not used). Thus, if this process were put in parallel with a process $\overline{a}\langle c\rangle.\,\overline{c}$, that first emits a name c along a, then emits on this channel c, two synchronisations could occur:

$$a(b).\, b \mid \overline{a}\langle c\rangle.\, \overline{c} \xrightarrow{\tau} c \mid \overline{c} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}$$

The previous transitions are instantiations of the more general law:

$$a(\widetilde{x}).\, P \mid \overline{a}\langle\widetilde{b}\rangle.\, Q \xrightarrow{\tau} P\{\widetilde{b}/\widetilde{x}\} \mid Q$$

($\{\widetilde{b}/\widetilde{x}\}$ denotes the syntactic replacement of $\widetilde{x}$ with $\widetilde{b}$, in a capture-avoiding way)

The fact that the exchanged names are the same used as guards presents a new problem: equations that may have seemed safe previously, now may become unguarded, as a consequence of some synchronisation. The equation $X = y.\, X \mid \overline{b}$ has a unique solution; however, the equation $X = a(y).\, (y.\, X \mid \overline{b}) \mid \overline{a}\langle b\rangle$ does not: after a first τ-step, it reduces to $X = b.\, X \mid \overline{b}$, and then another τ-step brings it to what is, essentially, the equation $X = X$.

The unique-solution property is also impacted by the way we choose to build a LTS to represent this behaviour. Indeed, we want to model the behaviour of a program even if no synchronisation is possible. We often use the rule

$$\frac{}{a(\widetilde{x}).\, P \xrightarrow{a(\widetilde{b})} P\{\widetilde{b}/\widetilde{x}\}}$$

where bound names $\widetilde{x}$ are instanciated by arbitrary names $\widetilde{b}$, as to model the fact that an external process may output any names along a.

In contrast, we sometimes require that in the previous rule, the names $\widetilde{b}$ are fresh, meaning that they do not appear free in P; we call the LTS thus defined the *ground* LTS. While the ground LTS is not adequate for the full π-calculus, it has a much simpler theory and accounts for the potential constraints we might impose to the language (hence constraining the type of names that processes may output). This is done by considering subcalculi of the π-calculus, where some constructs are constrained or simply absent from the language.

Congruence and up-to-context. In the full π-calculus, bisimilarity is not a congruence: this means that even though two processes P and Q might be bisimilar, when put in some common context C, the resulting processes $C[P]$ and $C[Q]$ might not be bisimilar anymore. Of course, up-to-context cannot be a sound technique in this case, as it implies congruence. To recover context-based techniques, an often used solution is to consider subcalculi of the π-calculus in which bisimilarity is a congruence.

There are many variants of the π-calculus in the litterature [SW01]. Among the calculi with the most robust theory, an impoortant one is the Asynchronous π-calculus (Aπ), in which communication is asynchronous, meaning it is not possible to know whether a message has been received yet, or if it the communication is still waiting to occur. This is done by forbidding that the output action prefixes any process other than the null process $\mathbf{0}$ (prefixed processes can be either $a(x).\, P$ for any P, or $\overline{a}\langle b\rangle.\, \mathbf{0}$). In other words, outputs always occur in parallel. In the asynchronous π-calculus, bisimilarity is indeed a congruence. Furthermore, we are allowed use the ground LTS for Aπ, which simplifies the theory of proof techniques. Another calculus for which bisimilarity is a congruence is the Internal π-calculus (Iπ), in

which communication always uses fresh names. This calculus has a theory very close to that of CCS, and thus bisimilarity is also a congruence. However, while the congruence property is essential to even consider context-based techniques, the converse is not true: if unique-solution techniques are always sound when congruence holds (at least for strong equivalences), this is not necessarily the case for up-to-context techniques.

An important difference between unique solution of equations and up-to techniques arises in the asynchronous π-calculus. In this setting, forms of bisimulation enhancements that involve 'up to context', such as 'up to expansion and context', require closure of the candidate relation under substitutions . It is an open problem whether this closure is necessary.

Higher-order calculi. The Higher-Order π-calculus is a higher-order concurrent language, whereby communication of channel names is not allowed, as processes themselves are exchanged along communications. For instance, the process $a(X).\,(X \mid \overline{b}\langle X\rangle)$ receives another process on a, and immediately runs it, while sending in parallel a copy of this process on b.

Up-to-context techniques can be particularly effective in such higher-order languages. Other notable higher-order languages include the λ-calculi, and the Ambients calculus (e.g., [Las98a, Las98b, KW06, SW01, MN05]). Unfortunately, in such calculi, proving the soundness of up-to context techniques can be surprisingly hard. Even in pure λ-calculi, and for the most basic form of bisimilarity (e.g., the call-by-name or call-by-value λ-calculus and Ambramsky's applicative bisimilarity [Abr87]) long-standing open problems remain about soundness, even though congruence is known to hold. Higher-order process calculi are the class of languages in which up-to context techniques in the literature are most scarce. Recently, techniques of this kind have been derived exploiting fully-abstract translations into first-order calculi (CCS-like or π-calculus-like) [MPS14]. However fully abstract translations are sensitive to the grammar of the language chosen: a modification to the grammar may break or prevent a fully abstract translation to be defined, or, at the very least, will require a careful re-examination of the full abstraction proof.

We thus look for direct proofs of soundness, that do not rely on translations into first-order languages. These are rare. The only such direct proofs are [MN05], for the Ambient calculus, and [SKS11] for the Higher-Order π-calculus (HOπ). The Ambient calculus represents a rather special case of higher-order calculus, for processes can move but cannot be communicated. Moving a process is quite different from communicating it as in HOπ: in the former case the process will always be run, immediately and exactly once; in the latter case, in contrast, the process may be copied, and it is the recipient of the process that decides when and where to run each copy. Thus the problems of soundness for up-to-context only show up in a limited form in Ambients. The up-to-context technique considered in [SKS11] is for environmental bisimilarity. This bisimilarity involves universal quantifications on processes supplied by the environment. Up-to-context is essential for limiting the burden due to such quantifications. The contexts used have constraints and are disallowed in certain clauses (this is necessary for the soundness of the technique).

Contributions

Contributions on the unique solution technique

The main contributions of this thesis follow two axis. We introduce in this section our contribution on the unique-solution technique. This work will be tested against an open problem: the problem of full abstraction for Milner's encoding of the call-by-value λ-calculus in the π-calculus. This is a well-known and challenging problem, that we use to test the effectiveness of the unique-solution techniques introduced in this thesis.

While our solution to this problem is enabled by our work on unique solution, these subjects are somewhat orthogonal, as full abstraction is an interesting problem by itself. We thus introduce it in the next section. A by-product of the study of full abstraction were a few further developments of the unique-solution technique; these are presented in the current section.

We study the technique of *unique solution of equations* for (weak) behavioural relations. We mainly focus on bisimilarity but we also consider other equivalences, such as trace equivalence, as well as preorders such as trace inclusion. We look for conditions, that, in conjunction with the syntactic guardedness condition of CCS, are sufficient to ensure that a given equation enjoys unique solution.

We draw inspiration from Roscoe's work on unique solution for CSP to formulate the counterpart of these results in the operational setting of CCS and bisimilarity. In comparison with the denotational CSP proof, the operational CCS proof is more complex. The operational setting offers however a few advantages. First, we can formulate more refined versions of the theorem, in which we distinguish between different forms of divergence. Notably, we can ignore divergences that appear after finite unfoldings of the equations, called *innocuous* divergences in this paper. (These refinements would look less natural in the denotational and trace-based setting of CSP, where any divergence causes a process to be considered undefined.) This allows us to establish some completeness results: in some restricted setting, equations with a unique solution are precisely equations with only innocuous divergences. However this does not hold in general: there are equations with a unique solution but no divergences, even in CCS. Equations of this type do not seem to appear in practice.

A second and more important advantage comes as a consequence of the flexibility of the operational approach: the unique-solution theorems can be tuned to other behavioural relations (both equivalences and preorders), and to other languages.

To highlight the latter aspect, we present abstract formulations of the theorems, on a generic LTS (i.e., without reference to CCS). In this abstract formulation, the body of an equation becomes a function on the states of the LTS. The theorems for CCS are instances of the abstract formulations. Similarly we can derive analogous theorems for other languages. We illustrate this flexibility using rule formats [BIM88]: we show that any language that enjoys congruence properties and that fit within certain standard rule formats, such as GSOS [BIM88] or tyft/tyxt [GV92], also enjoy our unique-solution theorem. We describe the abstract setting, and show how they illustrate the relationship between guardedness

and formats with lookaheads (tyft/tyxt). We then develop our own rule format, tailored specifically for our technique, inspired by rule formats for weak bisimilarity [vG05, vG11, UP02, Blo95], but much more general.

The abstract version of our main unique-solution theorem has been formalised using the Coq proof assistant [Dur17].

We develop several tools to facilitate the application of the technique in practice. In particular, one such result allows us to transplant uniqueness of solutions from a system of equations, for which divergences are easy to analyse, to another one. Another result is about the application of the technique to preorders. And lastly, we develop sufficient, decidable, conditions for non-divergence.

One of the most powerful up-to technique is the *'up to transitivity and context'* technique due to Pous, which relies on a termination hypothesis [Pou08] – which is reminiscent of our non-divergence hypothesis. Indeed we show that our unique-solution technique generalises 'up to expansion' and combines it with 'up to context' and 'up to transitivity'. We show that, modulo an additional hypothesis, our techniques are at least as powerful as this up-to technique: any up-to relation can be turned into a system of equations of the same size (where the size of a relation is the number of its pairs, and the size of a system of equations is the number of its equations) for which uniqueness of solutions holds.

The relationship between up-to-context techniques and unique solution of equation is less clear in languages that have name-passing or higher-order features; particularly, the soundness of up to context is an open problem for both the π-calculus and the Higher-Order π-calculus. This is even true of some of the well-behaved subcalculi of the π-calculus, such as Aπ, where, as far as we know, a closure by substitution of the candidate bisimulation up to is required. Our unique-solution techniques are strongly reminiscent of up to context techniques (the body of an equation acts like a context that is erased in a proof using 'up to context'); yet, surprisingly, we are able to port it to some languages with higher-order features, and to the π-calculus. We port it to the full π-calculus– baring that we have to consider closed abstractions only, and that divergences are much more common when the ground LTS is not available – as well as to several subcalculi of the π-calculus: Aπ, Iπ, and ALπ. Furthermore, we develop conditions for non-divergence for these calculi, enabled by the ground LTS.

We also transport the theory of unique solution to the setting of HOπ. We focus on *normal bisimilarity* [San96a] to analyse HOπ processes, for two main reasons. First, it is the most effective in proofs, as its clauses do not make use of additional universal quantifications with respect to ordinary bisimulation of CCS-like processes (other forms of bisimilarity, such as context bisimilarity or environmental bisimilarity, make use of universal quantifications in the input terms supplied by the external observer, as well as in other clauses). Second, precisely due to such lack of universal quantifications, the 'contextual' properties of normal bisimilarity are quite delicate. Even proving substitutivity with respect to basic operators such as parallel composition is hard (indeed, usually this is proved by relying on mappings onto other forms of bisimilarity or onto first-order calculi). No direct proofs of soundness of forms of up-to-contexts exists.

The structure of the proofs of the unique-solution theorems for HOπ or the π-calculi is similar to that of CCS. We consider this as a positive outcome: the main objective of this study was to examine if and how the CCS proofs could be transported onto a higher-order setting. In HOπ, there are however noticeable differences in the proofs. For instance, often the proofs require reasoning on sequences of transitions. Now, in CCS the derivative of any transition is a process. In HOπ, in contrast, the derivative of an input is an abstraction, that needs to receive a process before becoming a process itself; similarly for output transitions, whose derivatives are concretions. This has also consequences on the reasoning about unfolding of equations. Similar issues with instantiation of terms occur in the definitions of bisimilarity in CCS and in HOπ, and are at the heart of the differences between them.

We also discuss how we get our theorem for free in languages that enjoy first-order encodings [MPS14], further reinforcing the robustness and portability of our technique. And lastly, we also discuss the unique-solution technique in the λ-calculus. While the results in the λ-calculus are limited in scope (due to the omnipresence of divergences in its theory), these are not surprising: fully abstract translations of the λ-calculus in the π-calculus exist, that are the main subject of our studies on full abstraction.

Full abstraction for the encodings of the λ-calculus in the π-calculus

To test the effectiveness of the unique-solution technique introduced in the previous section, we use it to study two well-known problems from concurrency theory, one of which was still open.

Milner's work on functions as processes [Mil90a, Mil92], that shows how the evaluation strategies of *call-by-name λ-calculus* and *call-by-value λ-calculus* [Abr87, Plo75] can be faithfully mimicked in the π-calculus, is generally considered a landmark in Concurrency Theory, and more generally in Programming Language Theory. The comparison with the λ-calculus is a significant expressiveness test for the π-calculus. More than that, it promotes the π-calculus to be a basis for general-purpose programming languages in which communication is the fundamental computing primitive. From the λ-calculus point of view, the comparison provides the means to study λ-terms in contexts other than purely sequential ones, and with the instruments available to reason about processes. Further, Milner's work, and the works that followed it, have contributed to understanding and developing the theory of the π-calculus.

More precisely, Milner shows the operational correspondence between reductions in the λ-terms and in the encoding π-terms. He then uses the correspondence to prove that the encodings are *sound*, i.e., if the processes encoding two λ-terms are behaviourally equivalent, then the source λ-terms are also behaviourally equivalent in the λ-calculus. Milner also shows that the converse, *completeness*, fails, intuitively because the encodings allow one to test the λ-terms in all contexts of the π-calculus — more diverse than those of the λ-calculus.

The main problem that Milner work left open is the characterisation of the equivalence on λ-terms induced by the encoding, whereby two λ-terms are equal if their encodings are behaviourally equivalent π-calculus terms. The question is largely independent of the precise form of behavioural equivalence adopted in the π-calculus because the encodings are

deterministic (or at least confluent). We consider contextual equivalence (that coincides with may testing and trace equivalence) and barbed congruence (that coincides with bisimilarity).

For the call-by-name λ-calculus, the answer was found shortly after Milner's study [San93b, San00]: the equality induced is the equality of Levy-Longo Trees [Lon83], the lazy variant of Böhm Trees. It is actually also possible to obtain Böhm Trees, by modifying the call-by-name encoding so to allow also reductions underneath a λ-abstraction, and by including divergence among the observables [SX14]. These results show that, at least for call-by-name, the π-calculus encoding, while not fully abstract for the contextual equivalence of the λ-calculus, is in remarkable agreement with the theory of the λ-calculus: several well-known models of the λ-calculus yield Levy-Longo Trees or Böhm Trees as their induced equivalence [Lév75, Lon83, Bar84].

As an example of application of our techniques in the π-calculus, we revisit the completeness part of the proof of full abstraction for the encoding of the call-by-name λ-calculus into the π-calculus [San00, SX14] with respect to Levy-Longo Trees (LTs). The proof in [San00, SX14] uses 'up to expansion and context'. Such up-to techniques seem to be essential: without them, it would be hard even to define the bisimulation candidate. For our proof using unique-solution, there is one equation for each node of a given LT, describing the shape of such node.

For call-by-value, in contrast, the problem of identifying the equivalence induced by the encoding has remained open, for two main reasons. First, tree structures in call-by-value are less studied and less established than in call-by-name. Secondly, proving completeness of an encoding of λ into π requires sophisticated proof techniques. For call-by-name, for instance, a central role is played by *bisimulation up-to contexts*. For call-by-value, however, existing proof techniques, including 'up-to contexts', appeared not to be powerful enough.

One of the main contributions of this thesis is the study of the above open problem for call-by-value. Our main result is that the equivalence induced on λ-terms by their call-by-value encoding into the π-calculus is *eager normal-form bisimilarity* [Las05, LL05]. This is a tree structure for call-by-value, proposed by Lassen as the call-by-value counterpart of Levy-Longo Trees. Precisely we obtain the variant that is insensitive to some form of η-expansion, called *η-eager normal-form bisimilarity*. It validates the η-expansion law, when applied to variables:

$$\lambda y.\, xy = x \tag{2}$$

η-expansion is moreover always valid for abstractions, so that we have $\lambda y.\, (\lambda z.\, M)y = \lambda z.\, M$. However, in a weak call-by-value setting, η-expanded terms should not always be equated: indeed, Ω diverges, while $\lambda x.\, \Omega x$ converges to a value.

To obtain the results we have however to make a few adjustments to Milner's encoding and/or specialise the target language of the encoding. These adjustments have to do with the presence of free outputs (outputs of known names) in the encoding. Indeed, Milner's first encoding maps a λ-variable to a free output. We have, writing $\mathcal{V}[\![x]\!]\langle p\rangle$ for the encoding of the variable x, where p is the location at which x is evaluated:

$$\mathcal{V}[\![x]\!]\langle p\rangle \stackrel{\text{def}}{=} \overline{p}\langle x\rangle \ . \tag{3}$$

However this is troublesome for the validity of β_{v}-reduction (the property that λ-terms that are related by β_{v}-reduction — the call-by-value β-reduction — are are also equal in the π-calculus). Milner solved the problem by ruling out the initial free output $\overline{p}\langle x\rangle$ and replacing it with a bound output $\boldsymbol{\nu} y\, \overline{p}\langle y\rangle$ followed by a static link $y \blacktriangleright x$. A static link $y \blacktriangleright x$ forwards any name received by y to x, therefore acting as a substitution between x and y, but also constraining the context's access to x (the context may not observe inputs on x: it may only trigger outputs on x by performing an output on y). We write $\mathcal{V}'[\![\cdot]\!]$ for Milner's second encoding:

$$\mathcal{V}'[\![x]\!]\langle p\rangle \stackrel{\text{def}}{=} \boldsymbol{\nu} y\, (\overline{p}\langle y\rangle.\, y \blacktriangleright x) \ . \tag{4}$$

It was indeed shown later [San93a] that with (3) the validity of β_{v}-reduction fails. Accordingly, the final journal paper [Mil92] does not mention encoding (3). If one wants to maintain the simpler rule (3), then the validity of β_{v}-reduction can be regained by taking, as target language, a subset of the π-calculus in which only the output capability of names is communicated. This can be enforced either by imposing a behavioural type system including capabilities [PS96], or by working in a dialect of the π-calculus in which only the output capability of names is communicated, such as the Localised π-calculus [MS04].

The encoding (4) still makes use of free outputs — the final action of $y \blacktriangleright x$ is a free output on x. While this limited form of free output is harmless for the validity of β_{v}-reduction, we show that this brings problems when analysing λ-terms with free variables. Indeed, desirable call-by-value equalities fail; an example is given by the law:

$$I(xV) = xV \tag{5}$$

where I is $\lambda z.\, z$ and V is a value.

Law (5) is valid in any model of call-by-value, because any closing context equates these terms: for instance, $\lambda x.\, I(xV) = \lambda x.\, xV$.

Two possible solutions to recover law (5) are:

1. rule out the free outputs; this essentially means transplanting the encoding onto the Internal π-calculus [San96b], a version of the π-calculus in which any name emitted in an output is fresh;

2. control the use of capabilities in the π-calculus; for instance taking Asynchronous Local π [MS04] as the target of the translation. (Controlling capabilities allows one to impose a directionality on names, which, under certain technical conditions, may hide the identity of the emitted names.)

We consider both approaches, and show that in both cases, the equivalence induced coincides with η-eager normal-form bisimilarity.

In summary, our contributions on the call-by-value encoding are the following:

1. Showing that Milner's encoding fails to equate terms that should be equal in call-by-value.

2. Rectifying the encoding, by considering different target calculi, and investigating Milner's problem in such a setting.

The rectification we make does not really change the essence of the encoding – in one case, the encoding actually remains the same. Moreover, the languages used are well-known dialects of the π-calculus, studied in the literature for other reasons. In the encoding, they allow us to avoid certain accidental misuses of the names emitted in the communications. The calculi were not known at the time of Milner's paper [Mil92].

The unique-solution technique plays a central role in the proof. The structure induced by Milner's call-by-value encoding was expected to look like Lassen's trees; however existing proof techniques did not seem powerful enough to prove it. In this respect, another goal of this work is to carry out an extended case study on the applicability and expressiveness of our unique-solution technique.

Finally, we consider preorders — thus referring to the preorder on λ-terms induced by a behavioural preorder on their π-calculus encodings. We introduce a preorder on Lassen's trees (preorders had not been considered by Lassen) and show that this is the preorder on λ-terms induced by the call-by-value encoding, when the behavioural relation on π-calculus terms is the ordinary contextual preorder (again, with the caveat of points (1) and (2) above). With the move from equivalences to preorders, the overall structure of the proofs of our full abstraction results remains the same. However, the impact on the application of the unique-solution technique is substantial, because the phrasing of this technique in the cases of preorders and of equivalences is quite different.

Outline of the document

Prologue. In the prologue we introduce first the general framework for our techniques: CCS (0.1) and bisimulations (0.2). We present a quick introduction to up-to techniques (0.3), and a more detailed account of up-to-context techniques (0.4).

We then introduce the unique-solution technique, with Milner's unique solution Theorems [Mil89] (for weak and strong bisimilarity) in Section 0.5.

The motivation of the work presented here is to find adequate replacements for up-to-context techniques when we may not use them, and sometimes even proof techniques that are provably at least as powerful as up to context; a first illustration of such a correspondence is given by Sangiorgi's unique solution of contractions technique [San15], that we present in Section 0.6.

Chapter 1. The first chapter is dedicated to our 'unique solution of equations' technique, based on Roscoe's ideas [Ros92, Ros97], starting in Section 1.1 with CCS. We first give the statement and proof of the main theorem in CCS (1.1.1), and then discuss how to improve the result by taking into account non-innocuous divergences (1.1.2). Still in CCS, with present a few auxiliary results to facilitate the application of the technique (1.1.3), we compare it with other techniques in Section 1.1.4: first the 'unique solution of contractions' technique, then the most powerful up-to-context techniques available, for which we prove a completeness result. Lastly, we discuss completeness of the theorem with respect to equations enjoying uniqueness of solutions, for CCS (1.1.5): we show some partial results, as well as counter-examples for the general case.

In Section 1.2, we present our abstract framework for applying the technique; first for bisimilarity (1.2.1), then for other equivalences (1.2.2) and preorders (1.2.3).

In Section 1.3, we instantiate the results of the previous section with rule formats; standard ones, such as GSOS or the tyft/tyxt formats (1.3.1), or new formats we develop, tailored specifically for our technique (1.3.2).

Lastly, in Section 1.4, we adapt our technique to calculi with name-passing or higher-order features, such as the π-calculus (1.4.1.4) or the Higher-Order π-calculus (1.4.2.1). We give an example of a proof using our technique in the Higher-Order π-calculus (1.4.3).

Most of the results presented in Chapter 1 have been presented at the conference CONCUR'17 [DHS17] and later appeared, in long version, in [DHS19]. The study of unique solutions of equations for the Higher-Order π-calculus is presented in [DHS20].

Chapter 2. This chapter is dedicated to our work on full abstraction for Milner's encodings of the λ-calculus in the π-calculus [Mil92]. To introduce gently the reader into this matter, we start by revisiting, in the first subsection, the proof of full abstraction for the call-by-name λ-calculus. A similar proof already exists [San00], using up to context instead of unique solution of equations.

The second subsection discusses the call-by-value encoding. We first recall basic definitions about the call-by-value λ-calculus and its tree semantics in Section 2.2.1. Milner's

original encoding is presented in Section 2.2.2, while Section 2.2.3 presents our analysis of this encoding, beginning with the shortcomings related to the presence of free outputs. The first solution to these shortcomings is to move to the Internal π-calculus: this is described in Section 2.2.4. For the proof of completeness, in Section 2.2.7, we rely on unique solution of equations; we also compare such technique with the up-to techniques. The second solution is to move to the Asynchronous Local π-calculus: this is discussed in Section 2.2.8. We show in Section 2.2.9 how our results can be adapted to preorders and to contextual equivalence.

The results of Chapter 2 have been presented at the LICS conference [DHS18], in an abridged form.

Résumé en français

Cette thèse se place dans la tradition de l'étude de la théorie mathématique des langages de programmation, et plus précisément du développement de l'outillage sémantique pour leur étude. Afin de définir la sémantique d'un langage formel, une méthode éprouvée est la construction d'une *équivalence comportementale*: une telle équivalence permet d'identifier des programmes ayant le même *comportement* (c'est-à-dire qu'ils produisent les mêmes résultats, qu'ils accomplissent les mêmes communications, etc), et ce même si ce sont des programmes qui parviennent à ce résultat (ou à ces communications, dans le cas d'un langage concurrent) d'une manière tout à fait distincte.

Étudier de telles équivalences comportementales permet d'abstraire la syntaxe des langages, pour en extirper le contenu significatif, sémantique. Cependant, la concrétisation de cette opération par la *preuve de l'équivalence de deux programmes* peut être, dans certain cas, d'une extrême difficulté. Ainsi, si la définition de telles équivalences est aujourd'hui une question largement explorée et résolue, leur utilisation en pratique pose problème.

C'est pourquoi il est courant de faire appel à un outillage de *techniques de preuves*, permettant de faciliter la preuve que deux programmes ont bien le même comportement. De ce point de vue, la bisimilarité [Mil89] est une équivalence comportementale notable, car elle est définie par une telle technique de preuve: la méthode de la bisimulation, à qui elle doit dans une large mesure son succès – d'autant que cette méthode constitue aussi une technique de preuve pour d'autres équivalences. En effet, la bisimilarité est une équivalence plus *fine* que la plupart des autres équivalences considérées en pratique: pour montrer que deux programmes sont équivalents, il suffit souvent de montrer qu'ils sont bisimilaires (lorsque c'est possible). Les méthodes inspirées de la bisimilarité sont aujourd'hui utilisées dans de nombreux domaines de l'informatique, dans la théorie des automates par exemple [BP13, BP15b], ou encore, en pratique, dans la certification du compilateur CompCert [Ler].

Cette méthode est d'autant plus robuste qu'elle a été au fur et à mesure enrichie de diverses améliorations, dont entre autres les techniques dites « modulo » [Mil89, San98, PS11] (« up to » en anglais). Si la théorie de ces améliorations constitue un sujet d'étude à part entière, elle a également été comparée par Sangiorgi [San15] à la technique d'*unicité des solutions des équations*. Plus précisément, Sangiorgi établit une correspondance entre les techniques « modulo contextes » et la technique d'unicité des solutions. Cette technique propose d'utiliser une équation récursive entre programmes, dotée de la propriété d'*unicité des solutions*, caractérisant les équations dont tous les programmes qui sont solutions sont aussi équivalents entre eux (pour une équivalence comportementale donnée – ici, la bisimi-

larité). Pour prouver que deux programmes sont équivalents, il suffit alors de prouver qu'ils sont chacun solution de ladite équation; la preuve d'équivalence nécessaire pour le justifier étant souvent plus simple qu'une preuve d'équivalence directe entre les deux programmes.

Toute équation n'admet pas l'unicité des solutions: ainsi, l'équation $X = X$ admet tout programme comme solution. Pour utiliser une telle technique, il est donc vital de caractériser les équations admettant l'unicité des solutions, en proposant un ensemble de conditions suffisantes pour le garantir. Milner, dans son ouvrage pionnier sur la concurrence [Mil89], dans lequel il définit la bisimilarité, les techniques modulo, et les techniques d'unicité des solutions, donne un tel critère, qui s'appuie sur la notion syntaxique de « garde »: une équation est gardée si une opération doit nécessairement se produire avant toute récursion, c'est-à-dire avant que la variable d'équation n'apparaisse. Ainsi, l'équation $X = X$ n'est pas gardée, mais pour une opération $\mathtt{op}$, garantissant une forme de précédence, l'équation $X = \mathtt{op}(X)$ le serait. Cependant le critère proposé par Milner est contraignant, en ce qu'il interdit également toute forme de concurrence dans l'équation considérée. Cela est dû au fait que l'on considère des équivalences comportementales *faibles*: c'est-à-dire que l'on abstrait le programme du nombre de ses étapes de calcul internes (ou un protocole concurrent du nombre de synchronisations internes). considérant uniquement les opérations « visibles ». Or, dans ce cadre, le critère syntaxique habituel de garde ne garantit pas que des opérations *visibles* doivent se produire avant qu'apparaisse la variable récursive. Il est intéressant de constater que les techniques modulo également souffrent de limitations dans le cadre d'une équivalence faible, pour des raisons similaires.

Le langage CSP, proposé par Hoare [Hoa85], et dont on considère généralement une sémantique *dénotationnelle*, c'est-à-dire une sémantique mathématique abstraite, possède également une théorie de la récursion et d'unicité des solutions. En particulier, si les critères de Hoare sont soumis aux mêmes limitations que les critères de Milner, Roscoe [Ros92, Ros97] propose une garantie bien plus flexible pour garantir l'unicité des solutions pour une équivalence faible, reposant sur le concept de *divergence*: un programme diverge s'il a la possibilité de procéder à un nombre infini d'opérations internes, impossibles à constater pour un observateur extérieur.

Contributions

Les contributions de cette thèse s'articulent autour de deux axes: les apports sur les techniques d'unicité des équations, que nous présentons immédiatement, et l'utilisation de telles techniques dans l'étude d'une question ouverte, la question de « Full Abstraction » pour l'encodage de Milner du λ-calcul en appel par valeur vers le π-calcul [Mil92]. Si l'étude de cette question est rendue possible par la technique d'unicité des solutions, elle constitue en elle-même un sujet d'étude complexe et intéressant; ce problème est donc introduit séparément, dans la section suivante.

La principale contribution concernant la technique d'unicité des solutions s'appuie sur l'idée proposée par Roscoe, d'utiliser les divergences inhérentes à certaines équations pour discriminer certaines équations dotées de l'unicité des solutions. Le théorème de Roscoe

s'appuie sur la sémantique dénotationnelle de CSP; si cela permet une preuve plus simple et plus élégante (pour une sémantique adaptée, la preuve se résume à deux lignes), il est moins évident d'adapter le résultat à d'autres calculs ou à d'autres équivalences (tels que la bisimilarité). Nous proposons donc une preuve *opérationnelle*, c'est-à-dire s'appuyant sur les méthodes standard d'analyse de l'exécution pas à pas des programmes. Cette preuve, nous la réalisons d'abord dans le cadre de CCS [Mil89], un calcul de processus utilisé pour représenter des protocoles de communications statiques.

Cette approche admet plusieurs avantages: tout d'abord, cela nous permet de raffiner le résultat, en distinguant plusieurs types de divergences. En effet, nous pouvons ignorer certaines divergences « anodines », qui ne sont pas dues à la récursion. Cela nous permet d'établir des résultats de complétude: pour certains types d'équations (linéaires, entre autres), l'absence de divergences non anodines est équivalente à l'unicité des solutions.

De plus, cela permet d'établir un résultat de complétude vis-à-vis des techniques modulo, plus précisément de la technique « modulo contextes et transitivité » de Pous [Pou08], une des techniques modulo les plus puissantes: nous montrons que toute preuve réalisée avec cette technique modulo peut être réalisée avec notre technique d'unicité des solutions, et ce avec une preuve *de la même taille* (signifiant par là que la taille de la bisimulation est la même que celle du système d'équations).

Surtout, nous pouvons grâce à cette approche opérationnelle porter le résultat à d'autres calculs et d'autres équivalences. Nous proposons un cadre abstrait dans lequel le théorème est formulé, applicable à de nombreux langages. Les théorèmes pour CCS sont alors de simples instanciations de ces théorèmes abstraits. Pour illustrer la souplesse de ce cadre, nous étudions les formats « SOS » (pour « structural operational semantics »): la discipline SOS permet de décrire la sémantique formelle de langages; il est courant de considérer des restrictions de ces formats SOS, afin de décrire des familles de langages vérifiant certaines propriétés. Ainsi, tout langage décrit dans certains formats SOS peut instancier une variante de notre résultat. Nous montrons que c'est le cas pour des formats standards comme GSOS [BIM88] ou les formats tyft/tyxt [GV92], mais également pour un format plus général, taillé sur mesure pour notre technique. Les résultats dans le cadre abstrait sont formalisés grâce à l'assistant de preuve Coq [Dur17].

Nous portons également le résultat à des calculs d'ordre supérieur ou avec mobilité, où l'utilisation des techniques de preuves habituelles est plus délicate, voire impossible. De tels langages incluent le π-calcul, ou le π-calcul d'ordre supérieur, HOπ. Le π-calcul est un calcul de processus, similaire à CCS, mais dans lequel les processus sont *mobiles*, en ce que les communications permettent de transférer la connaissance des canaux de communication eux-mêmes, créant ainsi de nouvelles possibilités de communication. Le π-calcul d'ordre supérieur est similaire, mais les processus eux-mêmes peuvent être communiqués sur ces canaux de communication. Pour ces calculs, certaines questions liées à la validité des techniques modulo contextes sont toujours ouverte; de plus, il existe peu de preuves directes de validité pour de telles techniques pour des langages d'ordre supérieur.

Nous proposons aussi plusieurs exemples d'application, et plusieurs comparaisons avec d'autres techniques, notamment pour le π-calcul et HOπ.

Le problème de « Full Abstraction »

Afin de tester sérieusement les potentialités de notre technique, nous la confrontons à deux problèmes délicats, issus de la théorie de la concurrence pour processus mobiles entamée par Milner. Peu après l'introduction du π-calcul [MPW92, Mil99, SW01], Milner montre comment les deux principales stratégies d'évaluation du λ-calcul, l'appel par nom et l'appel par valeur, peuvent être fidèlement émulées par le π-calcul, sous la forme de deux encodages [Mil90a, Mil92]. Ce travail fondateur constitue non seulement un test d'expressivité pour le π-calcul, mais démontre une méthode permettant d'utiliser la communication comme primitive fondamentale du calcul. Cela fournit également un outil permettant d'étudier le λ-calcul dans un context concurrent, avec les outils disponibles dans ce cadre.

Milner établit plus précisément une correspondance opérationnelle, montrant que toute opération de réduction dans le λ-calcul peut être imité par le π-calcul. Il en déduit que les encodages sont *corrects*, c'est-à-dire que si deux termes sont équivalents en λ-calcul, leurs encodages sont équivalents en π-calcul. Cependant, les encodages ne sont pas *complets*: étant donné deux termes dont les encodages sont équivalents, on ne peut pas en déduire que les termes eux-mêmes le soient. Cela est lié aux contextes du π-calcul, qui sont dotés d'un pouvoir discriminant supérieur à ceux du λ-calcul, de par les primitives concurrentes du π-calcul, absentes en λ-calcul.

Milner pose donc la question de « Full Abstraction », c'est-à-dire de la détermination des équivalences (plus fines que les équivalences traditionnelles du λ-calcul) pour lesquelles la correspondance estparfaite; ou autrement dit, pour quelles équivalences les encodages sont-ils *corrects et complets*? La réponse, pour l'encodage en appel par nom, est apportée rapidement par Sangiorgi [San93a, San00]. Cette preuve recour, d'une manière qui semble indispensable, aux techniques modulo contextes; nous explorons à nouveau cette preuve, utilisant notre technique d'unicité des solutions.

Pour l'encodage en appel par valeur, par contre, le problème est resté ouvert. Cela est lié principalement: 1. aux équivalences pour le λ-calcul en appel par valeur, qui sont moins bien comprises; 2. à l'échec des techniques modulo pour cette preuve, dont les limitations (liées au fait que nous considérons des équivalences faibles) ne permettent pas de les appliquer alors. Une des principales contributions de cette thèse est précisément de proposer une solution à ce problème, s'appuyant sur l'unicité des solutions.

Afin d'arriver à une telle solution, une première étape consiste à reconsidérer l'encodage lui-même: en effet, pour l'encodage de Milner, l'égalité suivante

$$(\lambda y.\, y)(xV) = xV$$

où V est une valeur arbitraire, n'est pas valide. Du point de vue du λ-calcul, il s'agit pourtant d'une égalité essentielle, et il semble qu'elle devrait être valide en appel par valeur. Cela est dû principalement à une faiblesse dans l'encodage de la variable, donnant un pouvoir discriminant trop grand aux contextes, et qui empêchait déjà la correction pour une variante de l'encodage [San93a, SW01].

Nous considérons deux modifications de l'encodage, pour pallier ces problèmes: 1. modifier l'encodage, afin qu'il fasse partie d'un sous calcul du π-calcul, le π-calcul Interne; 2. ne

pas modifier l'encodage, mais contraindre les contextes, en limitant leur capacité d'émission (cela revient à restreindre les contextes à ceux du π-calcul Local).

Plus simplement, nous montrons que l'encodage de Milner échoue à identifier des termes qui devraient l'être, et nous proposons deux méthodes pour corriger cet encodage. Ces modifications sont mineures et ne modifient pas le principe de l'encodage. De plus, ces calculs n'existaient pas lorsque Milner a proposé ses encodages.

Nous montrons que l'encodage modifié est correct et complet pour une équivalence, la « *eager normal form bisimulation* », proposée par Lassen [Las05, LL05]. La technique d'unicité des solutions joue un rôle central dans la preuve. Cela nous permet aussi de développer des outils supplémentaires pour la technique d'unicité des solutions, qui facilitent son application. Nous considérons également la preuve pour des pré-ordres, ce qui est facilité par la souplesse de notre technique.

Contents

Prologue:
A survey of up-to-context & unique-solution techniques

General notations

Relations. We let letters $\mathcal{R}, \mathcal{S}$ range over relations. We use the infix notation for relations, e.g., $P\ \mathcal{R}\ Q$ means that $(P, Q) \in \mathcal{R}$, and we write $\mathcal{RS}$ for the composition of $\mathcal{R}$ and $\mathcal{S}$. A relation *terminates* if there is no infinite sequence $P_1\ \mathcal{R}\ P_2\ \mathcal{R}\ \ldots$.

Indexing sets and tuples. We use a tilde to denote a tuple, with countably many elements; thus the tuple may also be infinite. When the indexing set of a tuple, sequence or relation is not important, we omit it. We write $(P_i)_i$, Σ_i or $\cup_i$ for, respectively, the ordered tuple $\widetilde{P}$, the sum or the union, in place of $(P_i)_{i\in I}$, $\Sigma_{i\in I}$ and $\cup_{i\in I}$, omitting the indexing set I.

All relations are considered to be ordered; thus, a relation can be written as an ordered sequence of pairs $\{(P_i, Q_i)\}_i$, where (P_i, Q_i) is the i-th component of the relation.

All notations are extended to tuples componentwise; e.g., $\widetilde{P}\ \mathcal{R}\ \widetilde{Q}$ means that $P_i\ \mathcal{R}\ Q_i$, for each component i of the tuples $\widetilde{P}$ and $\widetilde{Q}$.

Definitions and abbreviations. We use the symbol $\stackrel{\text{def}}{=}$ for abbreviations; for instance, $P \stackrel{\text{def}}{=} G$, where G is some expression, means that P stands for the expression G. In contrast, symbol $\stackrel{\triangle}{=}$ is used for the definition of constants, and $=$ is used for equations and for either syntactic equality or α-equivalence.

Other notations. If $\leq$ is a preorder, then $\geq$ is its inverse. Given any set X, we write $\mathbf{id}$ for the identity function $X \to X$, $\mathbf{id} : x \mapsto x$.

α-conversion. Free names, closed terms and α-conversion are defined as usual [Bar84, HS86]. We adopt the usual "Barendregt convention". This will allow us to assume freshness of bound variables and names whenever needed. We write $\mathsf{fn}(X)$ for the free names of X.

Substitutions. Substitutions are defined as usual; following standard conventions, we write $X\{Y/x\}$ for the object X in which the free name x is substituted for the object Y (this is not syntactic replacement : bound names should not be substituted). We write σ, σ' for substitutions, and $P\sigma$ for P on which σ is inductively applied to all free names.

Labeled relations and diagrams. Labeled relations are sets of relations indexed by sets of labels; we often write $\xrightarrow{\mu}$ for such relations, where μ is the label and $\rightarrow$ the labeled relation.

Given two relations $\mathcal{R}, \mathcal{R}'$, and two labeled relations $\rightarrow, \twoheadrightarrow$, writing μ, μ' for labels of the former and λ, λ' for the labels of the latter, the following diagram means : for all x, y, x', μ such that $x \mathcal{R} y$ and $x \xrightarrow{\mu} x'$, there exists y', λ such that $y \xrightarrow{\lambda} y'$ and $x' \mathcal{R}' y'$.

$$\begin{array}{ccc} x & \mathcal{R} & y \\ {\scriptstyle\mu}\downarrow & & \vdots\,{\scriptstyle\lambda}\downarrow \\ x' & \mathcal{R}' & y' \end{array}$$

Symmetrically, whenever the dotted arrow is on the left, we mean: for all x, y, y', λ such that $x \mathcal{R} y$ and $y \xrightarrow{\lambda} y'$, there exists x', μ such that $x \xrightarrow{\mu} x'$ and $x' \mathcal{R}' y'$.

$$\begin{array}{ccc} x & \mathcal{R} & y \\ \vdots\,{\scriptstyle\mu}\downarrow & & {\scriptstyle\lambda}\downarrow \\ x' & \mathcal{R}' & y' \end{array}$$

When it is clear from the context which of x' and y' implies the existence of the other, we do not use dotted arrows, and the following diagram:

$$\begin{array}{ccc} x & \mathcal{R} & y \\ {\scriptstyle\mathcal{R}_1} & & {\scriptstyle\mathcal{R}_2} \\ x' & \mathcal{R}' & y' \end{array}$$

might mean either:

1. $\forall x, y, x'$ s.t $x \mathcal{R}_1 x' \exists y', y \mathcal{R}_2 y'$ and $x' \mathcal{R}' y'$
2. $\forall x, y, y'$ s.t $y \mathcal{R}_2 y' \exists x', x \mathcal{R}_1 x'$ and $x' \mathcal{R}' y'$

$$\texttt{sum}\ \frac{}{\Sigma_{i\in I}\mu_i.\,P_i \xrightarrow{\mu_i} P_i} \qquad \texttt{parL}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \texttt{comL}\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\texttt{res}\ \frac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu} a\, P \xrightarrow{\mu} \boldsymbol{\nu} a\, P'}\ \mu \neq a, \overline{a} \qquad \texttt{const}\ \frac{P \xrightarrow{\mu} P'}{K \xrightarrow{\mu} P'}\ \text{if } K \triangleq P$$

Figure 1: The LTS for CCS

0.1 The calculus of communicating systems

We assume an infinite set of *names* $a, b, \ldots$ and a set of *constant identifiers* (or simply *constants*) to write recursively defined processes. The special symbol τ does not occur in the names and in the constants. The class of the CCS processes is built from the operators of parallel composition, guarded sum, restriction, and constants, and the guard of a sum can be an input, an output, or a silent prefix:

$$P \ := \ P_1 \mid P_2 \ \Big|\ \Sigma_{i\in I}\mu_i.\,P_i \ \Big|\ \boldsymbol{\nu} a\, P \ \Big|\ K \qquad\qquad \mu \ := \ a \ \Big|\ \overline{a} \ \Big|\ \tau$$

where I is a countable indexing set. Sums are guarded to ensure that behavioural equivalences and preorders are substitutive. We write $\mathbf{0}$ when I is empty, and $P + Q$ for binary sums, with the understanding that, to fit the above grammar, P and Q should be sums of prefixed terms. Each constant K has a definition $K \triangleq P$. We usually omit trailing $\mathbf{0}$, e.g., writing $a \mid b$ for $a.\,\mathbf{0} \mid b.\,\mathbf{0}$. We write $\mu^n.\,P$ for P preceded by n μ-prefixes. The operational semantics is given by means of a Labelled Transition System (LTS), and is given in Figure 1 (the symmetric versions of the rules `parL` and `comL` have been omitted). The *immediate derivatives* of a process P are the elements of the set $\{P' \mid P \xrightarrow{\mu} P' \text{ for some } \mu\}$. In a transition $P \xrightarrow{\mu} P'$, we call P' the residual of the transition. We use ℓ to range over visible actions (i.e., inputs or outputs, excluding τ).

Some standard notations for transitions: $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\stackrel{\mu}{\Longrightarrow}$ is $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$ (the composition of the three relations). Moreover, $P \xrightarrow{\hat{\mu}} P'$ holds if $P \xrightarrow{\mu} P'$ or ($\mu = \tau$ and $P = P'$); similarly $P \stackrel{\hat{\mu}}{\Longrightarrow} P'$ holds if $P \stackrel{\mu}{\Longrightarrow} P'$ or ($\mu = \tau$ and $P = P'$). We write $P(\xrightarrow{\mu})^n P'$ if P can become P' after performing n μ-transitions. Finally, $P \xrightarrow{\mu}$ holds if there is P' with $P \xrightarrow{\mu} P'$, and similarly for other forms of transitions.

Replication. The replication of a process P, denoted $!P$, is part of the original syntax of CCS [Mil89]. It stands for an infinite parallel composition of copies of P, and its operational semantics is given by the rule

$$\frac{!P \mid P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

Replication can be encoded by using recursive constants: we use $K_{!P} \triangleq K_{!P} \mid P$ as a substitute for $!P$. Likewise, we use the constant $K_{!\mu.P} \triangleq \mu.\,(P \mid K_{!\mu.P})$ to encode guarded

replication $!\mu.\, P$. Thus, unless specified otherwise, replications $!P$ or $!\mu.\, P$ are solely abbreviations for said constants.

0.2 The bisimulation proof technique

0.2.1 Bisimulation games

Consider two processes, P and Q. We would like to compare them. More precisely, we would like to know if if they have the same transitions, both now and later. Therefore, we compare the transitions they can perform; and then compare the residual processes. As executions of processes may be infinite, we have to do this operation coinductively.

This boils down to a simple game: one process of the pair, say P, might challenge the other, Q, by performing a transition $P \xrightarrow{\mu} P'$; Q has to answer by a transition with the same label, $Q \xrightarrow{\mu} Q'$. The game continues with P' and Q', any of which may start a new challenge. This is the bisimulation game: we call a relation $\mathcal{R}$ a bisimulation when any pair in the relation follows this pattern, given by the diagram

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\mu}\downarrow & & \downarrow{\scriptstyle\mu} \\ P' & \mathcal{R} & Q' \end{array}$$

To prove two processes are equivalent, it is thus enough to exhibit a bisimulation relation by which they are related.

There are many different variants on the bisimulation game; we call *strong bisimulation* the previous one. For any bisimulation game, we can define the corresponding *bisimilarity*, i.e., the union of all bisimulation relations, or, equivalently, the largest one. Bisimilarities usually are equivalence relations, and thus *behavioural equivalences*. Bisimulation can be seen as a proof technique, used to prove that two processes are bisimilar.

Definition 0.2.1 (Strong bisimilarity)**.** A process relation $\mathcal{R}$ is a *strong bisimulation* if, whenever $P\ \mathcal{R}\ Q$, we have:

1. $P \xrightarrow{\mu} P'$ implies that there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P'\ \mathcal{R}\ Q'$;
2. the converse, on the transitions from Q.

P and Q are *strongly bisimilar*, written $P \sim Q$, if $P\ \mathcal{R}\ Q$ for some strong bisimulation $\mathcal{R}$.

Strong bisimilarity relates processes whose behaviour is similar, when taking also into account internal steps τ. In other words, two processes have to be as efficient to be in a strong bisimulation. While it can be useful, as a behavioural equivalence, it is inadequate: we would rather relate processes that have the same visible transitions (inputs and outputs),

but that are not necessarily as efficient, i.e., that differ in the amount of internal steps they perform.

Such an equivalence would be weak bisimilarity: it is based on a variant of the previous strong bisimulation game, but it is insensitive to efficiency, as it uses, instead of transitions $\xrightarrow{\mu}$ as challenges, weak transitions $\stackrel{\mu}{\Longrightarrow}$, that can contain any number of internal steps (before or after the possibly-visible action μ). As answers, it also allows transitions $\stackrel{\hat{\mu}}{\Longrightarrow}$ that are either weak or null (when $\mu = \tau$ only): if a process is only performing internal steps, the other is allowed to be more efficient and, as a response, perform no actions.

Definition 0.2.2 (Weak bisimilarity)**.** A process relation $\mathcal{R}$ is a *(weak) bisimulation* if, whenever $P \mathcal{R} Q$, we have:

1. $P \stackrel{\mu}{\Longrightarrow} P'$ implies that there is Q' such that $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$;
2. the converse, on the transitions from Q.

P and Q are *bisimilar*, written $P \approx Q$, if $P \mathcal{R} Q$ for some (weak) bisimulation $\mathcal{R}$.

As we are not interested in complexity or measures of efficiency, we focus on *weak* behavioural relations. Therefore, bisimilarity means weak bisimilarity by default, and bisimulation means weak bisimulation.

Weak bisimulations, as defined above, are not convenient when writing proofs: for each pair P, Q in $\mathcal{R}$, we have to consider all possible transitions $\stackrel{\mu}{\Longrightarrow}$ from P and Q. Because of this, the same transition has to be analysed several times.

Take, for instance, the processes $P = \tau.\tau.\overline{a}$ and $Q = \overline{a}$. To build a bisimulation containing P and Q, we have to consider the pairs (P, Q), $(\tau.\overline{a}, \overline{a})$, $(\overline{a}, \overline{a})$. For P, we have three transitions to consider. For instance, consider the transition $P \stackrel{\tau}{\Longrightarrow} \overline{a}$; we also have to break it down, considering first the transition $P \xrightarrow{\tau} \tau.\overline{a}$, and then the transition $\tau.\overline{a} \xrightarrow{\tau} \overline{a}$.

We therefore give a characterisation of weak bisimulations that is more useful for concrete proofs, where only strong transitions of a process have to be tested - but the challenged process might answer by weak or null transitions $\stackrel{\hat{\mu}}{\Longrightarrow}$.

Lemma 0.2.3 ([San11])**.** *A relation $\mathcal{R}$ is a weak bisimulation if and only if $P \mathcal{R} Q$ implies:*

1. *whenever $P \xrightarrow{\mu} P'$, there is Q' such that $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$;*
2. *the converse, on the transitions from Q.*

Proof. Because $\xrightarrow{\mu} \subseteq \stackrel{\mu}{\Longrightarrow}$, the first implication is immediate. Conversely, assume $\mathcal{R}$ is a relation such as described in the hypothesis; we have to show it is a weak bisimulation. Assume w.l.o.g. that $P \stackrel{\mu}{\Longrightarrow} P'$. We proceed by induction over the length of the transition. □

We often consider the characterisation in Lemma 0.2.3 to be the default bisimulation game for weak bisimilarity, and by (weak) bisimulation we mean this characterisation, rather than the game from Definition 0.2.2.

0.2.2 Congruence properties

Example 0.2.4 (Structural congruence)**.** Thanks to strong bisimilarity $\sim$, we can recover usual algebraic laws of CCS processes. These basic, syntactic laws, are often used to define the *structural congruence* [Mil89], an equivalence relation that contains all the following equalities:

	Strong bisimulation relation required to prove the law
$P \mid \mathbf{0} \sim P$	$\{(P \mid \mathbf{0}, P) \mid P \text{ is any process}\}$
$P \mid Q \sim Q \mid P$	$\{(P \mid Q, Q \mid P) \mid P, Q \text{ are any processes}\}$
$(P \mid Q) \mid R \sim P \mid (Q \mid R)$	$\{((P \mid Q) \mid R, P \mid (Q \mid R)) \mid \forall P, Q, R\}$
$\boldsymbol{\nu} x\,(\boldsymbol{\nu} y\,P) \sim \boldsymbol{\nu} y\,(\boldsymbol{\nu} x\,P)$	$\{(\boldsymbol{\nu} x\,(\boldsymbol{\nu} y\,P), \boldsymbol{\nu} y\,(\boldsymbol{\nu} x\,P)) \mid \forall P, x, y\}$
$\boldsymbol{\nu} x\,\mathbf{0} \sim \mathbf{0}$	$\{(\boldsymbol{\nu} x\,\mathbf{0}, \mathbf{0}) \mid \forall x\}$
$\boldsymbol{\nu} x\,(P \mid Q) \sim \boldsymbol{\nu} x\,P \mid Q$ if $x \notin \mathsf{fn}(Q)$.	$\{(\boldsymbol{\nu} x\,(P \mid Q), \boldsymbol{\nu} x\,P \mid Q) \mid \forall P, Q, x,\ x \notin \mathsf{fn}(Q)\}$
$!P \mid P \sim !P$	(Discussed in Section 0.3.)

Structural congruence is required to be an equivalence relation that contains previous equalities, and that also verifies the *congruence* property (hence its name). This means that, when two processes P and Q are related by structural congruence, they can be freely substituted by each other, as components of a larger process: assume process R contains P in its syntax tree; then write R' for R when the subterm corresponding to P has been replaced by Q. When P and Q are related, so are required to be R and R'.

In general, we define the congruence property by using the notion of *context*, i.e., processes that may contain one or several occurrences of a hole $[\cdot]$, acting as a place holder for processes. We also consider contexts with finitely many holes $[\cdot_1], \ldots, [\cdot_n]$, acting as place holders for multiple processes.

We write $C[P]$, where P is a process and C a context, for the process equal to C where every occurrence of the hole $[\cdot]$ has been simultaneously replaced by P. Likewise, for a context C with multiple holes and a list of processes $\widetilde{P}$, we write $C[\widetilde{P}]$ for C where every occurrence of every hole $[\cdot_i]$ is simultaneously replaced by the corresponding process P_i.

Definition 0.2.5 (Congruence)**.** A relation $\mathcal{R}$ is a congruence if $P\ \mathcal{R}\ Q$ implies, for all context C, $C[P]\ \mathcal{R}\ C[Q]$.

We write $\mathcal{C}(\mathcal{R})$ for the closure of a relation under contexts:

$$\mathcal{C}(\mathcal{R}) \triangleq \{(C[P], C[Q]) \mid P\ \mathcal{R}\ Q,\ C \text{ is an arbitrary context}\}.$$

A relation is a congruence if $\mathcal{C}(\mathcal{R}) \subseteq \mathcal{R}$.

Congruence is a highly desirable property for a behavioural equivalence, as it allows compositional reasoning: to show that $P \mid Q \approx P' \mid Q$, it is sufficient to show that $P \approx P'$, assuming $\approx$ is a congruence.

Milner [Mil89] shows that strong and weak bisimilarities are indeed congruences.

Theorem 0.2.6 ([Mil89]). *In CCS, strong and weak bisimilarities are congruences.*

Congruence is a property of relations over processes, but it depends on the language: the same equivalence, say strong or weak bisimilarity, might be a congruence or not, depending on whether some behaviours or some constructs are part of a language. Example 0.2.7 below illustrates this for CCS.

Structural congruence is usually the finest congruence relation that we bother to consider.

Example 0.2.7 (Unguarded sums and congruence). In the original presentation of CCS [Mil89], Milner proposes the unguarded sum construct $+$, replacing guarded sums. The process $P+Q$ might perform the same transitions as the processes P and Q, reducing to the same residual:

$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \frac{Q \xrightarrow{\mu} Q'}{P + Q \xrightarrow{\mu} Q'}$$

However, the presence of the unguarded sum construct prevents bisimilarity from being a congruence: for any process P, $\tau.P \approx P$, but in general $\tau.P + \tau.Q$ is not equivalent to $P + Q$. Take, for instance, $\tau.\overline{a} + \tau.\overline{b}$ and $\overline{a} + \overline{b}$. There is a transition $\tau.\overline{a} + \tau.\overline{b} \xrightarrow{\tau} \overline{a}$, but there is no action $\overline{a} + \overline{b}$ might perform to reduce to a process equivalent to $\overline{a}$.

Barbed congruence. While, for CCS, we are happy to use its LTS to define equivalences, this is not the case for higher-order calculi, such as the λ-calculus, or calculi with name-passing, such as the π-calculus. In these calculi, it is more natural to reason using internal transitions, but not a fully-fledged LTS. The standard method to define an equivalence for these calculi is to define a notion of *observable*, and deem two processes to be equivalent if they produce the same observables when placed in arbitrary contexts. This approach has the advantage that

1. we only reason about the immediate observables of the terms or processes (and not on its later behaviour)

2. they are congruences by construction.

In CCS, observables are the input or output actions a process may perform, possibly after some internal actions. The standard equivalence for CCS and the π-calculus is *barbed congruence* [MS92]: processes are equivalent if compared by being placed in an arbitrary context,

Bisimulations and bisimilarity are thus often used as proof techniques for reference equivalences, such as barbed congruence for CCS and the π-calculus, rather than serve as an equivalence on its own. In this spirit, we will aim for bisimilarity to coincide with *barbed congruence* along this thesis. Barbed congruence is similar to contextual equivalence, but allows for branching to different equivalence classes after some internal actions, in the spirit of bisimilarity.

Definition 0.2.8 (Barbed congruence). *Barbed bisimilarity* is the largest symmetric relation $\simeq^{\bullet}$ on CCS processes such that $P \simeq^{\bullet} Q$ implies:

1. If $P \to P'$ then there is Q' such that $Q \Rightarrow Q'$ and $P' \simeq^\bullet Q'$.

2. For all a, $P \stackrel{a}{\Rightarrow}$ iff $Q \stackrel{a}{\Rightarrow}$ and $P \stackrel{\bar{a}}{\Rightarrow}$ iff $Q \stackrel{\bar{a}}{\Rightarrow}$.

We say that P and Q are *barbed congruent*, written $P \simeq Q$, if for each context C, it holds that $C[A] \simeq^\bullet C[B]$.

Milner and Sangiorgi show that, restricting CCS to its finitely branching fragment, barbed congruence and weak bisimilarity indeed coincide [MS92].

Proposition 0.2.9. *Barbed congruence $\simeq$ coincides with weak bisimilarity $\approx$ on image-finite CCS processes.*

Remark 0.2.10. Bisimulation can be used as a proof technique not only for barbed congruence, but for virtually any equivalence, as it is usually the finest equivalence we might consider – as long as in the considered language, bisimilarity is a congruence.

0.2.3 Non-branching equivalences

There are many behavioural equivalences, other than bisimilarity, for process calculi [vG90, vG93], distinct even for a language as simple as CCS. More complex languages might necessitate refinements of these various base equivalences, yielding even more equivalences (e.g., the π-calculus). However, bisimilarity can be used as a proof technique for most of these equivalences, as it is among the finest equivalences (albeit, as a proof technique, it is not necessarily complete, in the sense that it might not be powerful enough to prove any equality holding for this equivalence).

We illustrate this fact with one of the most important equivalences from concurrency theory, trace equivalence. Trace equivalence has been used to define several other well-known equivalences, among which the testing equivalence, the must and may equivalences. These trace-based equivalences, including trace equivalence itself, differ from bisimilarity most notably in the fact that they are non-branching, as illustrated by the example thereafter.

We give below the definition for the weak trace inclusion and weak trace equivalence (that we consider, as usual, as the defaults); strong trace inclusion and equivalence can be deduced from the following definition, replacing weak transitions $\stackrel{\mu}{\Rightarrow}$ with transitions $\stackrel{\mu}{\to}$.

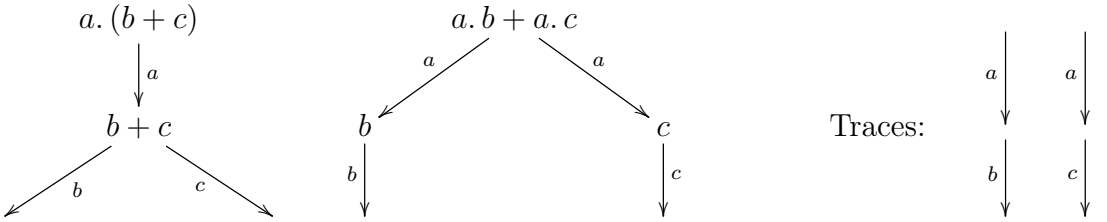
We call a finite sequence of actions $s = \mu_1, \ldots, \mu_n$, where each μ_i is a visible action, a *trace*. Accordingly, an infinite sequence of such actions $s = (\mu_i)_{i\in\mathbb{N}}$ is called an *infinite trace*. Given $s = \mu_1, \ldots, \mu_n$ a trace, we write $P \stackrel{s}{\Rightarrow}$ if $P \stackrel{\mu_1}{\Longrightarrow} P_1 \stackrel{\mu_2}{\Longrightarrow} P_2 \ldots P_{n-1} \stackrel{\mu_n}{\Longrightarrow} P_n$, for some processes $P_1, \ldots, P_n$. Likewise, given $s = (\mu_i)_{i\in\mathbb{N}}$ an infinite trace, we write $P \stackrel{s}{\Rightarrow}$ if there are processes $(P_i)_{i\in\mathbb{N}}$ such that $P = P_0$ and for all $i \in \mathbb{N}$, $P_i \stackrel{\mu_i}{\Longrightarrow} P_{i+1}$.

Definition 0.2.11 (Trace-based relations)**.** Two processes P, Q are in the *trace inclusion*, written $P \preceq_{\mathrm{tr}} Q$, if $P \stackrel{s}{\Rightarrow}$ implies $Q \stackrel{s}{\Rightarrow}$, for each trace s. They are *trace equivalent*, written $P \approx_{tr} Q$, if both $P \preceq_{\mathrm{tr}} Q$ and $Q \preceq_{\mathrm{tr}} P$ hold.

Two states P, Q are in the *infinite trace inclusion*, written $P \subseteq_{tr^\infty} Q$, if $P \stackrel{s}{\Rightarrow}$ implies $Q \stackrel{s}{\Rightarrow}$, for each finite or infinite trace s. They are *infinite trace equivalent*, written $P \approx_{tr}^{\infty} Q$, if both $P \subseteq_{tr^\infty} Q$ and $Q \subseteq_{tr^\infty} P$ hold.

Example 0.2.12 (Incompleteness of bisimilarity w.r.t trace equivalence)**.** We show that bisimilarity is strictly finer than trace inclusion. As this example does not use any τ-action, it is applies to strong and weak equivalences alike. Consider the processes $a.\,(b+c)$ and $a.\,b+a.\,c$. These theorems have the same traces, namely, a, ab, and ac. However, bisimilarity is branching, meaning it can detect forks in the execution of the process, while the traces only give information on the global sequence of actions that a process might perform.

Indeed, to show that $a.\,(b+c) \approx a.\,b+a.\,c$, and given that $a.\,b+a.\,c \xrightarrow{a} b$, we would have to exhibit a process P such that $a.\,(b+c) \xrightarrow{a} P$ and $P \approx b$; the only candidate for such a process would be $b+c$, however, $b+c \not\approx c$ (they may not produce the same visible action). Thus, there is no such P, and $a.\,(b+c) \not\approx a.\,b+a.\,c$.



0.3 Up-to techniques

0.3.1 Inadequacy of bisimulations

In this section we discuss examples illustrating the need for enhancements of the bisimulation proof method (both in the weak and the strong case): indeed, proving a relation is a bisimulation implies checking all transitions for each pair in the relation; hence, the largest the relation is, the more work is needed. Furthermore, to prove even basic laws, one may have to exhibit infinite and convoluted bisimulation relations, containing many redundant pairs.

Example 0.3.1 (Inadequacy of strong bisimulations)**.** Consider the law

$$!(a+b) \sim !a \mid !b \tag{6}$$

We search for a bisimulation candidate, $\mathcal{R}$, containing the pair $(!(a+b), !a \mid !b)$. As illustrated by the following diagram, this relation must related processes with an arbitrary number of residual null processes in parallel:

$$\begin{array}{ccc}
!(a+b) & \mathcal{R} & !a \mid !b \\
\downarrow{\scriptstyle a} & & \downarrow{\scriptstyle a} \\
!(a+b) \mid \mathbf{0} & \mathcal{R} & !a \mid \mathbf{0} \mid !b \\
\downarrow{\scriptstyle b} & & \downarrow{\scriptstyle b} \\
!(a+b) \mid \mathbf{0} \mid \mathbf{0} & \mathcal{R} & !a \mid \mathbf{0} \mid !b \mid \mathbf{0} \\
\ldots & & \ldots
\end{array}$$

Those null proccesses accumulate along the execution. We thus have to consider an infinite bisimulation relation, with pairs

$$!(a+b) \mid \overbrace{\mathbf{0} \mid \ldots \mid \mathbf{0}}^{n+m \text{ times}} \quad \mathcal{R} \quad !a \mid \overbrace{\mathbf{0} \mid \ldots \mid \mathbf{0}}^{n \text{ times}} \mid !b \mid \overbrace{\mid \mathbf{0} \mid \ldots \mid \mathbf{0}}^{m \text{ times}}$$

We already established associativity and commutativity of parallel composition, as well as deletion of null processes: $P \mid \mathbf{0} \approx P$ in Example 0.2.4 (algebraic laws of structural congruence). Using these laws, we may rewrite any of the previous pairs to the original pair $(!(a+b), !a \mid !b)$. It is thus desirable to reason up to these algebraic laws, and reuse the already-established laws for $\sim$ along the proof.

Law 6 can be generalized as follows, for any processes P and Q:

$$!(a.\, P + b.\, Q) \sim !a.\, P \mid !b.\, Q \tag{7}$$

Law 6 is then obtained from Law 7 by replacing P and Q with the null process. The new candidate bisimulation should factor, not only the residual P and Q processes, replacing the null processes of Law 6, but also their derivatives. Assume, for instance, that $P \xrightarrow{\mu} P'$, we then have

$$\begin{array}{ccc}
!(a.\, P + b.\, Q) & \mathcal{R} & !a.\, P \mid !b.\, Q \\
\downarrow{\scriptstyle a} & & \downarrow{\scriptstyle a} \\
!(a.\, P + b.\, Q) \mid P & \mathcal{R} & !a.\, P \mid P \mid !b.\, Q \\
\downarrow{\scriptstyle \mu} & & \downarrow{\scriptstyle \mu} \\
!(a.\, P + b.\, Q) \mid P' & \mathcal{R} & !a.\, P \mid P' \mid !b.\, Q \\
\ldots & & \ldots
\end{array}$$

In this case, the bisimulation proof method demands that we explore the whole transition system emanating from P or Q. We can further illustrate the redundancy of the pairs contained within relation $\mathcal{R}$: consider the pair $(!(a.\, P + b.\, Q) \mid P, !a.\, P \mid !b.\, Q \mid P)$, obtained from $\mathcal{R}$ by rearranging parallel compositions. The processes it contains are processes already related by $\mathcal{R}$, but enriched with a common context $[\cdot] \mid P$.

For weak bisimulations, we need not only to account for redundancies, but also for the so-called 'administrative reductions', i.e., transitions $P \xrightarrow{\tau} P'$ where $P \approx P'$. An instance of such transitions is given by the law

$$\boldsymbol{\nu} a\, (a.\, P \mid \overline{a}.\, Q) \approx \boldsymbol{\nu} a\, (P \mid Q) \tag{8}$$

This law corresponds to a deterministic τ-transition, that has to occur before the process can proceed.

The problem raised by such 'administrative reductions' when writing bisimulation proofs is best illustrated by the following example.

Example 0.3.2 (Inadequacy of weak bisimulations)**.** We consider the equality

$$!a.\, b \mid !\overline{a} \approx !a \mid !b \mid !\overline{a} \tag{9}$$

It holds because any input on b performed by the right-hand process can be reached by the left-hand process, by performing an administrative reduction $!a.\, b \mid !\overline{a} \xrightarrow{\tau} b \mid !a.\, b \mid !\overline{a} \mid \mathbf{0}$. However, there is no limit to the amount of administrative reductions both of these processes are able to perform before any visible transition occurs, creating many redundant pairs, similar to Example 0.3.1.

Example 0.3.3 (Inadequacy of weak bisimulations, again)**.** We use two recursive constants to define an implementation of a simple counter, $\mathbf{C}$, containing a positive integer. The prefix i acts as an increment of the counter, whereas the prefix d represents the decrement; hence, the counter may only perform the action d as many times as it has already performed the action i. To store the value of the counter, $\mathbf{C}$ uses a private name a. Hence, $\mathbf{C}$ is the name-restriction of another recursive constant, $\mathbf{C_a}$:

$$\mathbf{C_a} \triangleq i.\, \overline{a}.\, \mathbf{C_a} \tag{10}$$

$$\mathbf{C} \triangleq \boldsymbol{\nu} a\, (\mathbf{C_a} \mid !a.\, d) \tag{11}$$

A simpler implementation of this behaviour uses either replication, as the process $!i.\, d$, or, equivalently, a recursive constant, defined as follows

$$\mathbf{C}' \triangleq i.\, (\mathbf{C}' \mid d) \tag{12}$$

To verify the behaviour of $\mathbf{C}$, we want to show it is equivalent to $\mathbf{C}'$. However, as in the previous examples, the bisimulation is infinite, the pairs contain an arbitrary number of m and $\mathbf{0}$ processes in parallel, and as in Example 0.3.2, the administrative reduction $\boldsymbol{\nu} a\, (\overline{a}.\, \mathbf{C_a} \mid !a.\, d) \xrightarrow{\tau} \boldsymbol{\nu} a\, (\mathbf{C_a} \mid !a.\, d \mid d)$ is not performed.

0.3.2 Refining the bisimulation proof technique

As illustrated by Examples 0.3.1, 0.3.2 and 0.3.3, while we are incentivized to reduce the size of the relations we consider, the smallest bisimulation containing a given pair is often infinite. However, we do not need to exhibit a full bisimulation relation: an *incomplete* bisimulation, i.e., a relation that is shown to be included in a bisimulation, would suffice.

Milner [Mil89] proposes a technique based on this idea: given a relation $\mathcal{R}$, the condition that the residual processes must be related by $\mathcal{R}$ itself is relaxed, as they only have to be related by a relation $f(\mathcal{R})$, for some function f over relations. Usually, the function f enlarges relation $\mathcal{R}$, by adding some redundant pairs to it, so that the condition is easier to check. We call such a relation a *bisimulation up to* f, and it can be described by the following game (for weak bisimulations):

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\mu}\downarrow & & \Downarrow{\scriptstyle\hat{\mu}} \\ P' & f(\mathcal{R}) & Q' \end{array}$$

Similarly, for the strong bisimulation game, we get the following diagram:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\mu}\downarrow & & \downarrow{\scriptstyle\mu} \\ P' & f(\mathcal{R}) & Q' \end{array}$$

When f is the identity, we get the usual bisimulation proof technique. We say of such a function f it is a *sound* up to technique if any relation $\mathcal{R}$ that is a bisimulation up to f is such that $\mathcal{R} \subseteq \approx$ (resp. $\mathcal{R} \subseteq \sim$ for strong bisimulations). Thus, if the up-to f technique is sound, one may prove that two processes are bisimilar by exhibiting a bisimulation up to f relating said processes.

We now fix a function f over the set of relations on processes.

Definition 0.3.4 (Bisimulation up to)**.** A relation R is a bisimulation up to f (resp. strong bisimulation up to f) if, whenever $P \mathcal{R} Q$, for all μ we have:

1. $P \xrightarrow{\mu} P'$ implies that there is Q' such that $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $P'\ f(\mathcal{R})\ Q'$;
   (resp. $P \xrightarrow{\mu} P'$ implies that there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P'\ f(\mathcal{R})\ Q'$;)
2. the converse, on the transitions from Q.

When a relation verifies the hypothesis of a bisimulation, except that the target relation is replaced by a different relation, say $\mathcal{S}$, we sometimes say that $\mathcal{R}$ progresses to $\mathcal{S}$. In the definition above, we would say that $\mathcal{R}$ progresses to $f(\mathcal{R})$.

Definition 0.3.5 (Sound up-to technique)**.** We say f is a *sound* up-to technique for bisimulation (resp. strong bisimulation) if, whenever $\mathcal{R}$ is a bisimulation up to f (resp. strong bisimulation up to f), then $\mathcal{R} \subseteq \approx$ (resp. $\mathcal{R} \subseteq \sim$).

When the bisimulation game (strong or weak) is clear from the context, we simply say that f is sound.

A first example of a sound up-to technique is the function $\mathcal{R} \mapsto \sim\mathcal{R}\sim$; this functions allows to rewrite the residual processes with any law valid for $\sim$. In the literature, this technique is often called 'up to $\sim$' [Mil89, SW01], and its corresponding bisimulations are called bisimulations up to $\sim$.

Theorem 0.3.6 ([Mil89])**.** *The function $\mathcal{R} \mapsto \sim\mathcal{R}\sim$ is a sound up-to technique for strong bisimulations.*

With that technique in our toolbox, we can treat the first equality proof from Example 0.3.1.

Example 0.3.7 (Example 0.3.1)**.** We recall Law 6 from Example 0.3.1:

$$!(a+b) \sim !a \mid !b$$

To show the validity of this law, we exhibit a strong bisimulation up to $\sim$; according to Theorem 0.3.6, this is a sound up-to technique. We show the singleton relation $\mathcal{R} \stackrel{\triangle}{=} \{(!(a+b), !a \mid !b\}$ is a strong bisimulation up to $\sim$. Consider, for instance, the transition along a:

$$\begin{array}{ccccccccc} !(a+b) & & & & \mathcal{R} & & & & !a \mid !b \\ {\scriptstyle a}\downarrow & & & & & & & & \downarrow{\scriptstyle a} \\ !(a+b) \mid \mathbf{0} & \sim & !(a+b) & & \mathcal{R} & & !a \mid !b & \sim & !a \mid \mathbf{0} \mid !b \end{array}$$

We only use laws from Example 0.2.4 (structural congruence), to rearrange parallel compositions (through associativity and commutativity), and delete $\mathbf{0}$ processes.

Example 0.3.8 (Up to bisimilarity and up to identity)**.** The functions $\mathcal{R} \mapsto \mathcal{R} \cup \{(P, P) \mid P \text{ is any process}\}$ and $\mathcal{R} \mapsto \mathcal{R} \cup \approx$ (resp. $\mathcal{R} \mapsto \mathcal{R} \cup \sim$ for strong bisimulations) are sound up-to techniques, standard in the litterature [Mil89, PS11, SW01]. The latter is strictly more general than the former. These basic techniques are, by construction, sound, including for other bisimulation games. Indeed, if $\mathcal{R} \cup \approx \subseteq \approx$, then $\mathcal{R} \subseteq \approx$ as well.

Fix some P; we show the equality $\tau.P \approx P$, by showing the singleton relation $\mathcal{R} \stackrel{\triangle}{=} \{(\tau.P, P)\}$ is a bisimulation up to identity. There are two types of transitions to consider: either $\tau.P \xrightarrow{\tau} P$, or there is a transition $P \xrightarrow{\mu} P'$ (in which case $\tau.P \xrightarrow{\tau} \xrightarrow{\mu} P'$).

$$\begin{array}{ccccc} \tau.P & \mathcal{R} & P & \qquad & \tau.P & \mathcal{R} & P \\ {\scriptstyle \tau}\downarrow & & \| & & {\scriptstyle \mu}\Downarrow & & \downarrow{\scriptstyle \mu} \\ P & = & P & & P' & = & P' \end{array}$$

0.3.3 Up-to techniques for weak bisimilarity

0.3.3.1 Up to bisimilarity

The original weak bisimulation proof method (Definition 0.2.2) is a standard ('symmetrical') bisimulation game, but played on a different LTS $\stackrel{\hat{\mu}}{\Longrightarrow}$. In particular, the up-to-bisimilarity technique is sound for this game, just as the up-to $\sim$ technique is sound for the strong bisimulation game. However, the same is not true of the bisimulation game defined in Lemma 0.2.3: the up-to-$\approx$ technique is, in this case, unsound [SM92, SR11]. We illustrate this with the singleton relation $\mathcal{R} \stackrel{\triangle}{=} \{(\tau.a, \mathbf{0})\}$. Although $\tau.a \not\approx \mathbf{0}$, $\mathcal{R}$ is a bisimulation up to $\approx$:

$$\begin{array}{ccccccc} & & \tau.a & \mathcal{R} & \mathbf{0} & & \\ & {\scriptstyle \tau}\swarrow & & & \| & & \\ a & \approx & \tau.a & \mathcal{R} & \mathbf{0} & \approx & \mathbf{0} \end{array}$$

The up-to bisimilarity technique is an essential tool for a bisimulation theory, as illustrated by Examples 0.3.2 and 0.3.3, and by Example 0.4.4 below. However, for weak

bisimulations, it is not available, contrary to strong bisimulations. Several techniques have been proposed to replace this technique. In this section, we discuss some techniques often used in place of up to bisimilarity.

One of the goals of this thesis is to find an adequate substitute for up to bisimilarity. We present in Chapter 1 a new technique, that replaces the up-to bisimilarity technique in most cases.

Remark 0.3.9 (Strong bisimilarity up to transitivity)**.** The up-to transitivity technique for strong bisimulations, used in [Pou08, SR11], allows one to use the relation itself to expand the target relation; we thus demand that a bisimulation up to transitivity progresses to $\mathcal{R}^*$. This is, in spirit, very similar to the corresponding up-to bisimilarity technique (up-to $\approx$ or up-to $\sim$). Thus, weak bisimilarity up to transitivity is not sound, for the same reasons bisimilarity up to bisimilarity is not sound.

0.3.3.2 Up to strong bisimilarity

The up-to $\sim$ technique is sound for weak bisimulations. Indeed, the problem discussed above is only possible because of a cycle in which τ-transitions are added, which is impossible with strong bisimilarity. However, this solution is not satisfying, as it severely limits the available operations and laws. If the structural congruence laws are still allowed, there are useful algebraic laws that are only valid for weak bisimilarity, such as law 8, that we recall here:

$$\boldsymbol{\nu} a\,(a.\,P \mid \overline{a}.\,Q) \approx \boldsymbol{\nu} a\,(P \mid Q) \tag{8}$$

Up to $\sim$ is, however, sufficient to treat Example 0.3.2.

Example 0.3.10 ([SR11])**.** Setting $P := !a.\,b \mid !\overline{a}$ and $Q := !a \mid !b \mid !\overline{a}$, we show that the following relation is a bisimulation up to $\sim$

$$\mathcal{R} := \{(P \mid b^n, Q) \mid n \in \mathbb{N}\}$$

- For the challenges from $P \mid b^n$: any transition from $P \mid b^n$ can be written as one of the following

$$P \mid b^n \xrightarrow{\tau} \sim P \mid b^{n+1} \qquad P \mid b^n \xrightarrow{a} \sim P \mid b^{n+1}$$
$$P \mid b^{n+1} \xrightarrow{b} \sim P \mid b^n \qquad P \mid b^n \xrightarrow{\overline{a}} \sim P \mid b^n$$

  Therefore we check that $Q \xrightarrow{\mu} \sim Q$ ($\mu \in \{a, \tau, \overline{a}, b\}$).

- If $Q \xrightarrow{\mu} Q'$, then $Q' \sim Q$. We can match such a transition with above transitions, except if $\mu = b$ and $n = 0$; there we have $P \xrightarrow{\tau}\xrightarrow{b} \sim P$.

0.3.3.3 Up to expansion

A better solution is proposed by Milner and Sangiorgi [SM92]. It is the up-to expansion technique, that allows some of the behaviour of weak bisimilarity (such as Law 8), while controlling some of the inefficiencies to prevent the degenerate cycles that break the up-to $\approx$ technique.

The Expansion pre-order. We define the expansion preorder, written $\preceq$, where $P \preceq Q$ intuitively means that P and Q have the same behaviour, and that P may not be 'slower' (in the sense of doing more $\xrightarrow{\tau}$ transitions) than process Q.

Definition 0.3.11. A process relation $\mathcal{R}$ is a bisimulation expansion if, whenever $P \mathcal{R} Q$, we have:

1. $P \xrightarrow{\mu} P'$, implies there exists some Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$;
2. $Q \xrightarrow{\mu} Q'$ implies there exists some P' such that $P \xrightarrow{\hat{\mu}} P'$ and $P' \mathcal{R} Q'$.

The expansion preorder, written $\preceq$, is the union of all bisimulation expansions. The converse of $\preceq$ is written $\succeq$.

Law 8 can be turned into an expansion:

$$\boldsymbol{\nu} a\,(a.\,P \mid \overline{a}.\,Q) \succeq \boldsymbol{\nu} a\,(P \mid Q) \tag{13}$$

We can use the expansion preorder as a sound up-to technique, by defining the corresponding function $\mathcal{R} \mapsto \succeq\ \mathcal{R}\ \preceq$. We can, however, improve this technique by making it asymmetrical: indeed, we only to verify this efficiency constraint on the playing process; we can rewrite the answering process with $\approx$. This is illustrated by the following bisimulation game:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\mu}\downarrow & & \Downarrow{\scriptstyle\hat{\mu}} \\ P' & \succeq \mathcal{R} \approx & Q' \end{array} \qquad\qquad \begin{array}{ccc} P & \mathcal{R} & Q \\ \Downarrow{\scriptstyle\hat{\mu}} & & {\scriptstyle\mu}\downarrow \\ P' & \approx \mathcal{R} \succeq & Q' \end{array}$$

Note that this technique does not fit in the framework provided by Definition 0.3.4, because it is in essence symmetrical. It does, however, fit within the more advanced lattice-theoretic framework provided in [San98, San95, Pou07, Pou08, PS11, Pou16].

0.4 Up-to-context techniques

0.4.1 The up-to-context technique in CCS

As illustrated by Example 0.3.1, specifically the proof of the equality

$$!(a.\,P + b.\,Q) \sim !a.\,P \mid !b.\,Q \tag{7}$$

we might need to erase a common context that the derivatives of the related processes share. This is done by using an up-to technique, expanding the bisimulation relation by closing it under all contexts. This technique is called the 'up to context' technique, and it uses the following function:

$$\mathcal{C} : \mathcal{R} \mapsto \{(C[P], C[Q]) \mid P \,\mathcal{R}\, Q,\ C \text{ is any context}\}$$

We want to show it yields a valid up-to technique for bisimilarity, i.e. that the following diagram is enough to deduce that $\mathcal{R}$ is contained in bisimilarity.

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\mu}\downarrow & & \Downarrow{\scriptstyle\hat{\mu}} \\ P' & \mathcal{C}(\mathcal{R}) & Q' \end{array}$$

In the case of CCS, this technique has been shown to be sound for both strong and weak bisimulation [San98] when it was first introduced. Remark that such context-based techniques are language-dependent, and cannot be proven sound for a bisimulation game, regardless of the language; to prove the soundness of such a technique, every constructor has to be considered. For instance, the unguarded sum operator breaks congruence for weak bisimilarity (see Example 0.2.7); but congruence is a precondition for up to context to be sound (the soundness of up to context implies congruence). Thus, the up-to-context technique is sound for CCS with guarded sums, but not for CCS with unguarded sums.

Example 0.4.1. (Examples 0.3.1, 0.3.7) We illustrate the use of up-to-context techniques with a first part of the proof of bisimilarity $!(a.\,P + b.\,Q) \sim\, !a.\,P \mid\, !b.\,Q$.

Indeed, consider the b-transition from either $!(a.\,P + b.\,Q)$ or $!a.\,P \mid\, !b.\,Q$; both residual processes are obtained by simply adding a common process Q in parallel. Thus, with the context $C = [\cdot] \mid Q$ and the singleton relation relating the considered processes, we can play a bisimulation up to context

$$\begin{array}{ccc} !(a.\,P + b.\,Q) & \mathcal{R} & !a.\,P \mid\, !b.\,Q \\ {\scriptstyle b}\downarrow & & \downarrow{\scriptstyle b} \\ !(a.\,P + b.\,Q) \mid Q & \mathcal{C}(\mathcal{R}) & !a.\,P \mid\, !b.\,Q \mid Q \\ {\scriptstyle !(a.P+b.Q)} & {\scriptstyle \mathcal{R}} & {\scriptstyle !a.P \mid !b.Q} \end{array}$$

The previous method, however, does not work for the a-transition as the processes we obtain are

$$!(a.\,P + b.\,Q) \mid P \text{ and } !a.\,P \mid P \mid\, !b.\,Q$$

These processes do not directly share a common context, some rearrangement of parallel composition is needed first. Although this makes it impossible to use up-to-context, we would only need to combine it with some of the previous up-to technique, such as up to $\sim$ or $\succeq$; even up to structural congruence would be sufficient, as illustrated below.

$$\begin{array}{ccccccc} !(a.\,P + b.\,Q) & & \mathcal{R} & & & & !a.\,P \mid !b.\,Q \\ \downarrow{\scriptstyle a} & & & & & & \downarrow{\scriptstyle a} \\ !(a.\,P + b.\,Q) \mid P & & \mathcal{C}(\mathcal{R}) & !a.\,P \mid !b.\,Q \mid P & \sim & & !a.\,P \mid P \mid !b.\,Q \\ {\scriptstyle !(a.P+b.Q)} & & {\scriptstyle \mathcal{R}} & {\scriptstyle !a.P \mid !b.Q} & & & \end{array}$$

(with the context $C = [\cdot] \mid P$)

Quite surprisingly, it is still unknown whether the congruence of bisimilarity implies the soundness of the up-to-context technique for bisimulation (this is also the case for strong bisimilarity). Proofs of congruence are often very similar to proofs of soundness of up to context; however, there are languages for which congruence is known, but the soundness of up to context is still an open question (see Open Problem 0.4.13).

Open problem 0.4.2. In a language $\mathcal{L}$ for which bisimilarity is a congruence, is up to context always a sound technique?

0.4.2 Combining up-to-context with other techniques

0.4.2.1 Up to expansion and contexts

As illustrated by Example 0.4.1 it is often desirable to combine up-to techniques, and in particular up-to-context techniques with up-to techniques allowing to manipulate and rewrite the residuals, such as up to $\sim$, up to $\approx$, or up to $\succeq$. For instance, Sangiorgi shows that up to $\succeq$ and context, obtained by combining up to context and up to $\succeq$, is indeed sound.

Theorem 0.4.3 ([SM92, SW01, San15])**.** *Up to $\succeq$ and context is a sound technique.*

Other common up-to techniques can be combined with up to context: for instance, for strong bisimulations and weak bisimulations alike, up to context and $\sim$ is a sound technique [San98]. We illustrate the usefulness of the up-to-context-and-expansion technique by revisiting Example 0.3.3

Example 0.4.4 (Example 0.3.3)**.** We use a bisimulation up to expansion and contexts to show that both counters $\mathbf{C}$ and $\mathbf{C}'$ are bisimilar;

The transition $\boldsymbol{\nu} a\,(\overline{a}.\,\mathbf{C_a} \mid !a.\,d) \xrightarrow{\tau} \boldsymbol{\nu} a\,(\mathbf{C_a} \mid !a.\,d \mid d)$ is an administrative reduction (meaning it is a deterministic τ-transition), hence $\boldsymbol{\nu} a\,(\overline{a}.\,\mathbf{C_a} \mid !a.\,d) \succeq \boldsymbol{\nu} a\,(\mathbf{C_a} \mid !a.\,d \mid d)$. Using the structural congruence equations (Example 0.2.4), we also get $\boldsymbol{\nu} a\,(\mathbf{C_a} \mid !a.\,d \mid d) \sim \boldsymbol{\nu} a\,(\mathbf{C_a} \mid !a.\,d) \mid d = \mathbf{C} \mid d$; we thus deduce $\boldsymbol{\nu} a\,(\overline{a}.\,\mathbf{C_a} \mid !a.\,d) \succeq \mathbf{C} \mid d$ (as $\sim \subseteq \succeq$).

We now show that the singleton relation $\mathcal{R} = \{(\mathbf{C}, \mathbf{C}')\}$ is a bisimulation up to expansion and contexts, and deduce that $\mathbf{C} \approx \mathbf{C}'$:

$$\begin{array}{ccccccc} \mathbf{C} & & & & \mathcal{R} & & \mathbf{C}' \\ \downarrow{\scriptstyle i} & & & & & & \downarrow{\scriptstyle i} \\ \boldsymbol{\nu} a\,(\overline{a}.\,\mathbf{C_a} \mid !a.\,d) & \succeq & \mathbf{C} \mid d & & \mathcal{C}(\mathcal{R}) & & \mathbf{C}' \mid d \end{array}$$

0.4.2.2 Combining up-to techniques

Several frameworks have been proposed to provide simple ways to combine up-to techniques, effectively generating up-to techniques for free: any technique obtained by combining other techniques fitting one such framework is immediately sound.

Sangiorgi first proposes the 'respectful functions' and 'compatible functions' frameworks [San98, San95, SW01]; the latter was then refined by Pous [Pou07, Pou08, PS11], and latter generalized further, with a description of the most general theory [Pou16], based on the concept of 'companion', encompassing both the concepts of respectfulness and compatibility.

These frameworks approach coinduction and up-to techniques from a lattice-theoretic point of view; a precise understanding of these notions is not needed here, and we refer the reader to the papers above for more details. Rather, the important aspect of these frameworks, is that they provide a very powerful tool for obtaining up-to techniques, more robust than a proof of soundness: for instance, functions that are 'compatible' do not only have sound corresponding techniques, but any combination of compatible functions is also compatible.

In the case of CCS, most techniques exhibited in the previous sections are either respectful, compatible, or at least 'below the companion', meaning they can be combined with other similar techniques. This is the case for up to context and up to $\sim$ in both the strong and weak cases, and of up to $\succeq$ in the weak case, as well as many other such techniques.

The exact terminology is not needed to understand the examples in this chapter; simply, when we say of a technique that it is respectful, compatible, or below the companion, it means it can be combined with other techniques easily. We also provide counter-examples, techniques that cannot be combined with other 'compatible' techniques, for instance, and thus which are not compatible themselves.

0.4.2.3 Sound but not compatible up-to-context techniques

Whether the congruence property implies the soundness of up to context is still an open question (see Open Problem 0.4.2); one might reasonably hypothesize that congruence could imply not only the soundness of up to context, but also its compatibility. However this is not the case: there are many situations where bisimilarity is a congruence, but $\mathcal{C}$, while a sound technique, is impossible to combine with other techniques (and hence not compatible, respectful, nor below the companion).

We illustrate the difficulty of combining up-to-context techniques with other useful, and otherwise sound (and compatible) up-to techniques. We give two known examples, from the literature (see [San98] and [PS11]).

Up to context and weak bisimilarity. The symmetrical weak bisimulation (Definition 0.2.2) game is a game for which both up to context and up to $\approx$ are sound techniques; but while up-to-contexts and up-to-$\sim$ can be combined in the strong case, and up-to-contexts with up-to-$\succeq$ in the weak case, we show that up to $\approx$ and contexts is not a sound technique [SM92].

A common counter-example for that technique is given by the relation $\mathcal{R} \stackrel{\triangle}{=} \{(b.\,a, b)\}$, which is a bisimulation up to $\approx$ and contexts, although $b.\,a \not\approx b$.

Let $C = \boldsymbol{\nu} b\,(\overline{b} \mid \cdot)$. Then, for all P, $C[b.\,P] \sim \tau.\,P \approx P$. Therefore $C[b.\,a] \approx a$ and $C[b] \approx \mathbf{0}$. We can then play the following bisimulation up to:

$$\begin{array}{ccccccc} b.\,a & & & & \mathcal{R} & & b \\ \Big\Downarrow{\scriptstyle b} & & & & & & \Big\Downarrow{\scriptstyle b} \\ a & \approx & \boldsymbol{\nu} b\,(\overline{b} \mid b.\,a) & \mathcal{C}(\mathcal{R}) & \boldsymbol{\nu} b\,(\overline{b} \mid b.\,\mathbf{0}) & \approx & \mathbf{0} \\ & & {\scriptstyle b.a} & {\scriptstyle \mathcal{R}} & {\scriptstyle b} & & \end{array}$$

This proof is reminiscent of the problem with the equation $X = b.\,\boldsymbol{\nu} b\,(\overline{b} \mid X)$, that uses the context $b.\,C$ as its body – corresponding to the first action b follow by the context C deleted by the use of the technique. This context is erased infinitely many times through the use of up to context, while the equation can be unfolded any number of time, creating only τ-transitions.

'Look-aheads'. We now illustrate the problem of combining up-to-context with other techniques in the most simple case of strong bisimulation. We exhibit a small toy language for which contexts are congruences, and up-to-context is sound. The language we consider only needs a CCS-like prefix construct, a base process (for instance, $\mathbf{0}$), and another construct that performs what we call a 'look-ahead ':

$$P \;:=\; \texttt{forward}(P) \;\mid\; a.\,P \;\mid\; \mathbf{0}$$

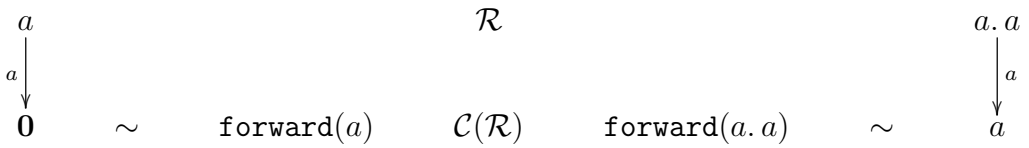
$a.\,P$ is the CCS-like prefix (with only one possible channel), $\mathbf{0}$ is the inactive process and `forward` is an operator whose behaviour is given by the rule

$$\frac{P \xrightarrow{a} Q \quad Q \xrightarrow{a} R}{\texttt{forward}(P) \xrightarrow{a} R}$$

Thus, the next transition of a process $\texttt{forward}(P)$ depends on the next two transitions of P. This is often called a 'look-ahead': when an operator of the language allows to 'see in the future', e.g., when its behaviour depends not on the immediate transitions of its arguments, but on their future transitions as well.

Bisimilarity is a congruence for this language (it can be defined within the framework of the so-called 'tyft format', ensuring that contexts preserve bisimilarity; more detailed are given about this format in Sections 0.4.3.1 and 1.3.1.

We show that, on this language, the function $\mathcal{C}$ is not compatible, respectful, or below any compatible function: the up-to-$\sim$ technique is compatible, however combining $\mathcal{C}$ and $\sim$ yields an unsound technique, allowing to prove incorrect bisimilarities. Consider processes a and $a.\,a$: they are not bisimilar. However we can show that $\mathcal{R} \stackrel{\triangle}{=} \{(a, a.\,a)\}$ is a bisimulation up to contexts and strong bisimulation. There is only two transitions to consider, the a transition from either a or $a.\,a$. Either way, $\mathcal{R}$ progresses to $\sim \mathcal{R} \sim$:



Therefore, the 'up to context and strong bisimulation' technique is not sound. Up-to-$\sim$ technique is, however, compatible. Hence, $\mathcal{C}$ is neither compatible nor below a compatible function.

The counterexample above does not show that $\mathcal{C}$ itself is not sound. As indeed, it is sound (see [PS11]). However, it does illustrate how limited in scope is the 'up to context' technique when it is sound but not compatible: it cannot be combined with even elementary up-to techniques.

0.4.3 Formats

Rule formats [AFV01, MRG07] are a means to enforce structural properties languages; these provide a specification for the form of the Structural Operational Semantics (SOS) rules used to describe the constructs of a language. In this section we give basic definitions for rule formats, as well as a few examples relevant to up-to-context techniques. For a complete overview of rule formats and for the relevant definitions, we refer the reader to [AFV01, MRG07, BIM88].

We consider a term algebra on a signature Σ. We write t, u for terms, $x, y, \cdots \in V$ for the variables and $\mu \in \Lambda$ for labels in the set of labels Λ.

Definition 0.4.5 (TSS)**.** Let Σ be a signature. A positive Σ-literal is an expression $t \xrightarrow{\mu} t'$ and a negative Σ-literal an expression $t \not\xrightarrow{\mu}$ with $t, t' \in \mathbb{T}(\Sigma)$ and μ an action. A transition rule over Σ is an expression of the form

$$\frac{H}{\alpha}$$

with H a set of Σ-literals (the premises of the rule) and α a positive Σ-literal (the conclusion). The left- and right-hand side of α are called the source and the target of the rule, respectively. A transition system specification (TSS), written (Σ, R), consists of a signature Σ and a collection R of transition rules over Σ.

0.4.3.1 Formats for strong bisimilarity

The GSOS format. One of the most common formats is GSOS [BIM88, vG05, FvG16]. This format is a generalization of the De Simone format, and is known to ensure that bisimilarity is a congruence.

Definition 0.4.6. A GSOS rule is a transition rule such that

- its source has the form $f(x_1, \ldots, x_{ar(f)})$ with $f \in \Sigma$ and $x_i \in V$,
- the left-hand sides of its premises are variables x_i with $1 \leq i \leq ar(f)$,

- the right-hand sides of its positive premises are variables that that are all distinct, and that do not occur in its source,
- its target only contains variables that also occur in its source or premises.

Furthermore, a positive GSOS rule is a rule whose premises are positive.

A GSOS language, or TSS in GSOS format, is a TSS whose rules are GSOS rules. A positive GSOS language is a language whose rules are in the positive GSOS format. In other words, if $\mathtt{op}$ is a construct of a GSOS language, its rules are in the format

$$\frac{\{x_i \xrightarrow{\mu_{i,j}} y_{i,j} \mid i \in I,\ 1 \leq j \leq m_i\} \cup \{x_j \xrightarrow{\mu'_{j,k}} \not\rightarrow \mid j \in J,\ 1 \leq k \leq n_j\}}{\mathtt{op}(\widetilde{x}) \xrightarrow{\mu} t}$$

where I, J are fixed indexing sets, and t may only use the variables x_i for $i \in I \cup J$ or $y_{i,j}$ for $i \in I$ and $1 \leq j \leq m_i$.

Corollary 0.4.7. *Strong bisimilarity is a congruence for any language in the GSOS format.*

Aceto et al. show that up to context is compatible for any language in a positive GSOS format.

Proposition 0.4.8 ([AFV01, BPPR17])**.** *Up to context is compatible for strong bisimulations for any positive GSOS language.*

Remark 0.4.9. Language in the GSOS format do not ensure that weak bisimilarity is a congruence: indeed, CCS with the sum operator (Example 0.2.7) is in the GSOS format, however bisimilarity is not a congruence.

0.4.3.2 Format for weak bisimilarity

Rule formats for weak bisimilarity are trickier. The 'WB cool GSOS' rule format, for weak bisimilarity, and other 'cool GSOS' formats, for variants of weak bisimilarity, are proposed in [Blo95, UP02], and further studied in [vG05, vG11]. They are shown to guarantee that bisimilarity is a congruence. These formats are complex; we refer the reader to [vG05, vG11] for more details. We only give the central idea of these formats, as we the format we develop in Section 1.3.2 is based on this same idea.

These formats crucially rely on the necessity for some of the operators of the language to have 'patience rules'; this enforces that these operators propagate τ-transitions. Only operators that somehow 'depend' on their arguments share this constraint.

A patience rule for the ith argument of an operator $\mathtt{op}$ of arity n (with $i \leq n$) is a rule of the shape:

$$\frac{x_i \xrightarrow{\tau} y}{\mathtt{op}(\widetilde{x}) \xrightarrow{\tau} \mathtt{op}(\widetilde{x}\{x_i/y\})}$$

These formats enforce that, whenever a variable appears both in a premise and in the target or source of a same rule, for a given operator $\mathtt{op}$, then this operators $\mathtt{op}$ must have a patience rule for this variable. The 'WB cool GSOS' rule format was also shown to guarantee that up to context is compatible.

Theorem 0.4.10 ([BPPR17]). *For a simply WB cool GSOS language, up to context is a compatible technique for weak bisimilarity.*

0.4.4 Name-passing and the up-to-context technique

In calculi with name-passing, such as the π-calculus, whereby substitutions inherited from the context are propagated to sub-processes, the soundness of up-to-context techniques can become a very delicate problem.

In the the full π-calculus [SW01], bisimilarity is not a congruence, hence up to context cannot be sound. We thus illustrate the problems with up to context for a sub-calculus of the π-calculus, for which bisimilarity is a congruence: the asynchronous π-calculus (abbreviated Aπ). We give a quick overview of Aπ (detailed definitions and properties are given in Section 1.4). Its syntax is given by the grammar:

$$P, Q \;:=\; a(b).\, P \;\mid\; \overline{a}b \;\mid\; P \mid Q \;\mid\; \boldsymbol{\nu} a\, P \;\mid\; !P$$

The single prefix construct $\mu.\, P$ from CCS is replaced by one construct for the input $a(b).\, P$ that binds the name b in P, and a construct for the output $\overline{a}b$ ($\overline{a}b$ is usually a short cut for $\overline{a}b.\, \mathbf{0}$, this thus amounts to say that an output can only prefix the null process). When a prefix input and an output synchronize, the name bound in the input prefix is substituted by the name in object position from the output:

$$a(b).\, P \mid \overline{a}c \xrightarrow{\tau} P\{c/b\}.$$

Comparing to the case of CCS, a new obstacle gets in the way of proving that bisimilarity is a congruence: the name-binding input prefix has to respect bisimilarity. This implies that substitutions too have to respect bisimilarity: indeed, $\boldsymbol{\nu} a\, (a(b).\, P \mid \overline{a}c)$ is bisimilar to $P\{c/b\}$ (and strongly bisimilar to $\tau.\, P\{c/b\}$). Assuming $P \approx Q$, to show that $C[P] \approx C[Q]$ means also proving that $P\sigma \approx Q\sigma$ for any substitution σ. Hence, congruence depends on the substitutivity of bisimilarity, i.e., whether $P \approx Q$ implies $P\sigma \approx Q\sigma$ for an arbitrary substitution σ. This is indeed shown by Sangiorgi [ACS98, SW01].

Lemma 0.4.11 ([SW01]). *If $P \approx Q$ (resp. $P \sim Q$), then for any substitution σ, $P\sigma \approx Q\sigma$ (resp. $P\sigma \sim Q\sigma$).*

The substitutions here can be any arbitrary substitution, and are not limited to injective substitutions, or fresh-name substitutions. Congruence of strong and weak bisimilarity follows.

Theorem 0.4.12 ([SW01]). *In Aπ, $\sim$ and $\approx$ are congruences.*

The dependence of congruence on substitutivity is also found as a dependence of up-to-context techniques on up-to-substitution techniques – for soundness and for compatibility alike; consider the following segment of a bisimulation up to context:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \downarrow{\scriptstyle \mu} & & \downarrow{\scriptstyle \mu} \\ \boldsymbol{\nu} a\,(a(b).\,P' \mid \overline{a}c) & \mathcal{C}(\mathcal{R}) & \boldsymbol{\nu} a\,(a(b).\,Q' \mid \overline{a}c) \end{array}$$

As the removed context acts, effectively, as a substitution, this amounts to the following bisimulation up to substitution:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \downarrow{\scriptstyle \mu} & & \downarrow{\scriptstyle \mu} \\ \boldsymbol{\nu} a\,(a(b).\,P' \mid \overline{a}c) & \mathcal{R} & \boldsymbol{\nu} a\,(a(b).\,Q' \mid \overline{a}c) \\ \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} \\ P'\{c/b\} & \mathcal{R}\{c/b\} & Q'\{c/b\} \end{array}$$

However, it is not known whether up to context or up to substitution are sound techniques (whether for strong or weak bisimilarity); furthermore, these techniques are not easily combined with other techniques, are they are not compatible, respectful, nor below the companion.

Indeed, a proof due to Damien Pous shows that the corresponding up-to functions are not below the companion (hence they cannot be compatible or respectful). To our knowledge, this result has never been published, nor has a similar result; because it refers to theories that are out of the scope of this Chapter, we refer the reader to Appendix B.1, containing Damien Pous' proof [Pou17].

Thus, up to context is not compatible (or below a compatible function), even for strong bisimulations, and, to the best of our knowledge, it is an open question whether it is sound.

Open problem 0.4.13. Is up to context a sound up-to technique for strong bisimilarity in Aπ?

Weak bisimilarity. As this problem is open even for strong bisimulation, and it is also the case of weak bisimulations. The objections to compatibility also apply to weak bisimilarity, hence while up to context might be sound for weak bisimilarity, it is not compatible (or below the companion).

0.5 Milner's unique solution of equations (An alternative to up-to-context techniques)

In his book [Mil89], Milner proposes, as an alternative to up-to-context techniques, the unique solution of equations technique. This technique, while very usable for strong bisimilarity, is

severely lacking for bisimilarity: it is restricted to equations using parallel composition in a very specific fashion.

The relationship between techniques relying on unique solution of equations and up-to-context techniques was later illustrated by Sangiorgi [San15], as explained in Section 1.1.4.1.

Systems of equations.

We need variables to write equations. We use capital letters X, Y, Z for these variables and call them *equation variables*. The body of an equation is a CCS expression possibly containing equation variables. We use $E, E', \ldots$ to range over *equation expressions*; these are process expressions that may contain occurrences of variables; that is, the grammar for processes is extended with a production for variables.

Definition 0.5.1. Assume that, for each i of a countable indexing set I, we have a variable X_i, and an expression E_i, possibly containing some variables. Then $\{X_i = E_i\}_{i \in I}$ (sometimes written $\widetilde{X} = \widetilde{E}$) is a *system of equations*. (There is one equation for each variable X_i.)

In the equation $X = E[X]$, we sometimes call body of the equation the equation expression E (we use the same terminology for systems of equations).

$E[\widetilde{P}]$ is the process resulting from E by replacing each variable X_i with the process P_i, assuming $\widetilde{P}$ and $\widetilde{X}$ have the same length. (This is syntactic replacement.) The components of $\widetilde{P}$ need not be different from each other, while this must hold for the variables $\widetilde{X}$.

Definition 0.5.2. Suppose $\{X_i = E_i\}_{i \in I}$ is a system of equations. We say that:

- $\widetilde{P}$ is a *solution of the system of equations for* $\approx$ if for each i it holds that $P_i \approx E_i[\widetilde{P}]$.
- The system has *a unique solution for* $\approx$ if whenever $\widetilde{P}$ and $\widetilde{Q}$ are both solutions for $\approx$, then $\widetilde{P} \approx \widetilde{Q}$.

Examples of systems with a unique solution for $\approx$ are:

1. $X = a.\, X$
2. $X_1 = a.\, X_2$, $X_2 = b.\, X_1$

The unique solution of the system (1), modulo $\approx$, is the constant $K \stackrel{\triangle}{=} a.\, K$: for any other solution P we have $P \approx K$. The unique solution of (2), modulo $\approx$, is given by the constants K_1, K_2 with $K_1 \stackrel{\triangle}{=} a.\, K_2$ and $K_2 \stackrel{\triangle}{=} b.\, K_1$; again, for any other pair of solutions P_1, P_2 we have $K_1 \approx P_1$ and $K_2 \approx P_2$.

Examples of systems that do not have unique solution are:

1. $X = X$
2. $X = \tau.\, X$

3. $X = a \mid X$

All processes are solutions of (1) and (2); examples of solutions for (3) are K and $K \mid b$, for $K \stackrel{\triangle}{=} a.\, K$

Remark 0.5.3. To prove that two processes P and Q are equivalent using the unique solution proof technique, one has first to find an equation $X = E[X]$ of which both P and Q are solutions. Then, a sufficient condition for uniqueness of solutions makes it possible to deduce $P \approx Q$.

The unique-solution method is currently particularly used in combination with algebraic laws [Ros10, BBR10, GM14].

Definition 0.5.4. An equation expression E is

- *strongly guarded* if each occurrence of a variable in E is underneath a visible prefix;
- *(weakly) guarded* if each occurrence of a variable in E is underneath a prefix, visible or not;
- *sequential* if each occurrence of a variable in E is only appears underneath prefixes and sums .

We say that an equation satisfies one of the above properties when its body does. These notions are extended to systems of equations in a natural way: for instance, $\{X_i = E_i\}_{i \in I}$ is guarded if each expression E_i is (w.r.t. every variable that occurs in E_i).

In other words, if the system is sequential, then for every expression E_i, any sub-expression of E_i in which X_j appears, apart from X_j itself, is a sum (of prefixed terms). For instance,

- $X = \tau.\, X + \mu.\, \mathbf{0}$ is sequential but not strongly guarded, because the guarding prefix for the variable is not visible.
- $X = \ell.\, X \mid P$ is guarded but not sequential.
- $X = \ell.\, X + \tau.\, \boldsymbol{\nu} a\, (a.\, \overline{b} \mid a.\, \mathbf{0})$, as well as $X = \tau.\, (a.\, X + \tau.\, b.\, X + \tau)$ are both guarded and sequential.

Remark 0.5.5 (Recursive specifications in ACP)**.** Systems of equations are called *recursive specifications* in the literature related to ACP [BW90]. In that context, other notions of guardedness have been studied. A sufficient condition, more powerful than Milner's, was originally given by Baeten, Bergstra and Klop [BW90], where synchronisation is transformed into a visible action, which is then deleted through an explicit operator. An equation is then guarded if no such deletion operator appears in its body.

In some process calculi, such as ACP [BW90] and mCRL2 [GM14], guardedness is synonymous, for an equation, to having a unique solution: this follows the *Recursive specification principle* (RSP), which states that guarded recursive specifications are unique (while

the *Recursive definition principle* states that recursive specifications do have solutions). Increasingly general definitions of guardedness that make this principle sound are then studied. In this framework, our contribution can be seen as a new notion of guardedness, stronger that what is found in the literature, under which the recursive definition principle is sound for weak bisimilarity.

Theorem 0.5.6 (Unique solution of equations, [Mil89])**.** *A system of equations has unique solution in CCS:*

1. *for $\sim$, if it is weakly guarded*

2. *for $\approx$, if it is strongly guarded and sequential*

The proof for $\sim$ is immediate. On the other hand, the proof for $\approx$ exploits an invariance property on immediate transitions for strongly guarded and sequential expressions, and then extracts a bisimulation (up to bisimilarity) out of the solutions of the system.

Example 0.5.7 ([Mil89])**.** To see the need of the sequentiality condition, consider the equation

$$X = \boldsymbol{\nu} a\, (a.\, X \mid \overline{a}) \ ,$$

where the occurrence of X in the equation expression is strongly guarded but not sequential. Any process that does not use a is a solution.

This equation's body corresponds to the context used in the first counter-example of Section 0.4.2.3, illustrating that the up-to-bisimilarity technique is not sound.

Remark 0.5.8. In CCS, ν is not a binder; this allows us to write equations where local names are used outside their scopes (for instance, $X = a.\, \boldsymbol{\nu} a(\overline{a} \mid X)$), as there is no alpha-renaming. It would still be possible, when ν has to be considered a binder (in a higher-order setting, for example), to write similar equations in a parametric fashion: $X = (a)\ a.\, \boldsymbol{\nu} a\, (\overline{a} \mid X\langle a \rangle)$. Such an approach is adopted in Section 1.4.1.4, to handle name passing.

The unique-solution proof technique for $\sim$. Given two processes P and Q, and an equation E that has a unique solution for $\sim$, it suffices to show that $E[P] \sim E[Q]$ to conclude that $P \sim Q$. More generally, given a system of equations $\widetilde{E}$ and processes $\widetilde{P}$, $\widetilde{Q}$, if $\widetilde{E}$ has a 'unique solution', $\widetilde{P}$ is solution of $\widetilde{E}$, and $\widetilde{Q}$ as well, then $\widetilde{P} \sim \widetilde{Q}$, and in particular, $P_i \sim Q_i$ for all i.

For strong bisimilarity, this technique is adequate, substituting for up-to-context techniques: in most cases, the condition from Theorem 0.5.6 is sufficient to obtain that the considered equations have a unique solution.

Example 0.5.9 (Unique solution and up to context, Examples 0.3.1 , 0.3.7 and 0.4.1)**.** The up-to-$\sim$ and context proof of Example 0.3.1 can be turned into a proof using unique solution

of equation. For this, we first turn the processes from Examples 0.3.1, 0.3.7 and 0.4.1 into recursive constant definitions:

$$K = a.\,K + b.\,K \tag{14}$$
$$K' = a.\,(K' \mid P) + b.\,(K' \mid Q) \tag{15}$$

and, in turn, make the definitions of these constants into equations:

$$X = a.\,X + b.\,X \tag{14}$$
$$X = a.\,(X \mid P) + b.\,(X \mid Q) \tag{15}$$

Theorem 0.5.6 certifies that Equations 14 and 15 have a unique solution. The constants are solutions of the corresponding equations by construction. For instance, it is easy to show that $!(a.\,P + b.\,Q)$ is solution of Equation 15: the corresponding singleton relation progresses to equality. There are two transitions to consider, along a and along b; for the transition along a, we get the diagram

$$\begin{array}{ccc} !(a.\,P + b.\,Q) & \mathcal{R} & a.\,(!(a.\,P + b.\,Q) \mid P) + b.\,(!(a.\,P + b.\,Q) \mid Q) \\ \downarrow{\scriptstyle a} & & \downarrow{\scriptstyle a} \\ !(a.\,P + b.\,Q) \mid P & = & !(a.\,P + b.\,Q) \mid P \end{array}$$

and similarly for the transition along b. We can thus conclude that $K' \sim !(a.\,P + b.\,Q)$.

For weak bisimilarity, this technique can only apply to sequential equations: this is very limitating, as indeed equations from Examples 0.3.2 and 0.3.3 are not sequential.

0.6 Unique solution of contractions

Sangiorgi [San15] proposes an approach to expand the use of the unique solution technique, in the case of weak bisimilarity. For this purpose, Sangiorgi introduces the contraction preorder, $\succeq_c$, a refinement of the expansion preorder. Because contraction is sensitive to internal steps, just as expansion is, Sangiorgi uses it to replace the the sequentiality hypothesis; Sangiorgi is thus able to prove that all weakly guarded systems of contractions have a unique solution for bisimilarity. A contraction is a pre-equation of the form

$$X \geq E$$

(rather than an equation $X = E$) Are considered solutions of a contraction processes such that $P \succeq_c E[P]$.

By doing so, Sangiorgi also illustrates the correspondence between the unique solution of contractions technique and the 'up to contraction and context' technique.

Definition 0.6.1 (bisimulation contraction)**.** A process relation $\mathcal{R}$ is a bisimulation contraction if, whenever $P\,\mathcal{R}\,Q$,

1. $P \xrightarrow{\mu} P'$ implies there is Q' such that $Q \xrightarrow{\hat{\mu}} Q'$ and $P' \mathcal{R} Q'$;

2. $Q \xrightarrow{\mu} Q'$ implies there is P' such that $P \stackrel{\hat{\mu}}{\Longrightarrow} P'$ and $P' \approx Q'$.

The contraction preorder, written $\succeq_c$, is the union of all bisimulation contractions.

Intuitively, in the definition above, Q is capable of mimicking P's transition, at least as efficiently as P (with as much or less internal steps). The contrary, on the other hand, does not need to hold: as soon as a challenge comes from Q, P can be as inefficient as it needs, as P' and Q' need only to be related by $\approx$.

Contraction is coarser than the expansion relation $\succeq$ of Definition 0.3.11. Thus the 'up-to contraction and context' technique is a refinement of the 'up-to expansion and context' technique (the former captures a larger set of relations because bisimilarity contraction is coarser than expansion).

Sangiorgi shows that the contraction preorder is a congruence:

Theorem 0.6.2 ([San15])**.** *$\succeq_c$ is a congruence in CCS.*

The proof is similar to analogous proofs for bisimilarity and expansion.

Systems of contractions. Given a system of pre-equations $\widetilde{X} \geq \widetilde{E}$, we say of the corresponding system of contractions that

- $\widetilde{P}$ is a solution if $\widetilde{P} \succeq_c \widetilde{E}[\widetilde{P}]$;
- it has *a unique solution for* $\approx$ if whenever $\widetilde{P}$ and $\widetilde{Q}$ are both solutions then $\widetilde{P} \approx \widetilde{Q}$.

Sangiorgi shows that the sequentiality hypothesis of Theorem 0.5.6 can be dropped for contractions: any weakly guarded system of contractions has a unique solution for $\approx$. The guardedness condition is also weakened: prefixes are not required anymore to be visible, as the equations are required to be weakly guarded (τ prefixes are thus allowed).

Theorem 0.6.3 (unique solution of contractions for $\approx$)**.** *A system of weakly-guarded contractions has a unique solution for $\approx$.*

Assume $\widetilde{P}$ and $\widetilde{Q}$ are solutions of a system of weakly guarded contractions. Sangiorgi shows that the relation

$$\mathcal{R} \stackrel{\triangle}{=} \{(R, S) \mid R \approx C[\widetilde{P}], S \approx C[\widetilde{Q}] \text{ for some context } C\}.$$

is a bisimulation. The proof exploits the following lemma.

Lemma 0.6.4 ([San15])**.** *Suppose $\widetilde{P}$ and $\widetilde{Q}$ are solutions of a system of weakly-guarded contractions. For any context C, if $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, then there is a context C' such that $R \succeq_c C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\hat{\mu}}{\Longrightarrow} \approx C'[\widetilde{Q}]$.*

The proof proceeds by induction over the length of the transition $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, and exploits the congruence of $\succeq_c$.

Indeed, if $X \geq E$ is a contraction, and P is such that $P \succeq_c E[P]$, then also

$$P \succeq_c \overbrace{E[E[\dots E}^{n \text{ times}}[P]\dots]$$

for any n.

Assume $P \stackrel{\mu}{\Longrightarrow}$ is a transition of length n or less. If E is weakly guarded, the transition $P \stackrel{\mu}{\Longrightarrow}$ can be mimicked by the context $E[E[\dots[\cdot]\dots]$: each hole is underneath at least n prefixes, and cannot contribute to an action in the first n transitions.

Example 0.6.5. Consider the equation from Example 0.5.7. It does not have a unique solution for $\approx$. However, the following contraction now has a unique solution for $\approx$:

$$X \geq \boldsymbol{\nu} a\, (a.\, X \mid \overline{a})$$

Its solution are all diverging processes (processes are diverging if they can perform an infinite number of internal steps), and may not produce any visible action. For instance, the process $\tau^\infty \stackrel{\triangle}{=} \tau.\,\tau.\,\tau\dots$ is a solution (it can be described as the constant $K_{\tau^\infty} = \tau.\, K_{\tau^\infty}$).

Any such solutions are indeed bisimilar: any process that may not perform any visible action is bisimilar to the null process $\mathbf{0}$.

0.6.1 Correspondence of unique solution and up-to

Sangiorgi shows that the unique solution of contractions technique is computationally equivalent to the 'up-to $\succeq_c$ and context'; computationally equivalent means that for any proof using one of these two techniques, there is a proof *of the same size* using the other.

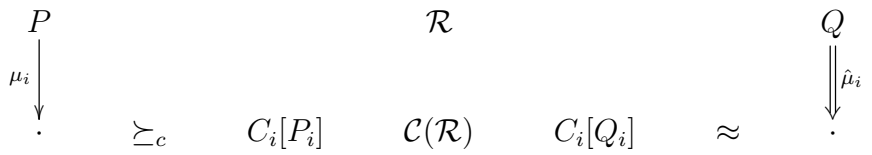
Theorem 0.6.6 ([San15])**.** *Suppose $\mathcal{R}$ is a bisimulation up-to $\succeq_c$ and context. Then there is a system of weakly-guarded contractions, of the same size as $\mathcal{R}$, of which $\mathcal{R}_1$ and $\mathcal{R}_2$ are solutions for $\succeq_c$.*

Conversely, suppose $\widetilde{P}$ and $\widetilde{Q}$ are solutions for $\succeq_c$ to the same system of weakly-guarded contractions. Then the relation $\{(P_i, Q_i)\}_i$ is a bisimulation up-to $\succeq_c$ and context.

The proof of the first part of the Theorem crucially relies on the so-called 'expansion law', adapted for contractions. This law states that a process P can be expressed as the sum of its immediate transitions associated with the remaining processes. In other words, if $P \xrightarrow{\mu_i} P_i$ is an enumeration of the transitions of P, we have

$$P \succeq_c \Sigma_i \mu_i.\, P_i$$

Sangiorgi uses this law, applied to a pair of processes P and Q related by a bisimulation up-to $\succeq_c$ and context $\mathcal{R}$, according to the following diagram



Considering also the transitions from Q, indexed by j, this immediately translates to the equation

$$X \succeq_c \Sigma_i \mu_i.\, C_i[X_i] + \Sigma_j \mu_j C_j[X_j]$$

where X_i are variables corresponding to the pairs $(P_i, Q_i) \in \mathcal{R}$, and similarly for X_j.

Remark 0.6.7. The 'expansion law', so crucial in the correspondence between the contraction technique and up-to techniques, is not valid asynchronous calculi, and particularly for the asynchronous π-calculus. Thus, any such perfect correspondence is unattainable. This explains why, while we manage to develop unique solution techniques for Aπ, as illustrated in Section 1.4, the mere soundness of up-to context for Aπ remains an open question (Section 0.4.4).

Having shown that these two techniques are equivalent, Sangiorgi derives the soundness of the up-to technique from the soundness of the contraction technique (i.e., Theorem 0.6.3).

Corollary 0.6.8 (soundness of 'bisimulation up-to $\succeq_c$ and context')**.** *If $\mathcal{R}$ is a bisimulation up-to $\succeq_c$ and context, then $R \subseteq \approx$.* □

Chapter 1

The unique solution technique

1.1 Unique solution in CCS

1.1.1 Divergences and Unique Solution

We introduce our proof technique in the setting of CCS, looking first at bisimilarity as a behavioural equivalence. We first introduce the concept of divergence, as it is used to discriminate equations that enjoy the unique-solution property from equations that do not.

Definition 1.1.1 (Divergence)**.** A process P *diverges* if it can perform an infinite sequence of internal moves, possibly after some visible ones; i.e., there are processes P_i, $i \geq 0$, and some n, such that $P = P_0 \xrightarrow{\mu_0} P_1 \xrightarrow{\mu_1} P_2 \xrightarrow{\mu_2} \dots$ and for all $i > n$, $\mu_i = \tau$. We call a *divergence of* P the sequence of transitions $\left(P_i \xrightarrow{\mu_i} P_{i+1}\right)_i$.

Example 1.1.2. The process $L \triangleq a.\,\boldsymbol{\nu} a\,(L \mid \overline{a})$ diverges, since $L \xrightarrow{a} \boldsymbol{\nu} a\,(L \mid \overline{a})$, and (leaving aside $\mathbf{0}$ and useless restrictions) $\boldsymbol{\nu} a\,(L \mid \overline{a})$ has a τ transition onto itself.

To define the divergences of an equation $X = E$, hence to ensure an equation has a unique solution, we need to reason with the *unfoldings* of said equation: we define the n-th unfolding of E to be E^n; thus E^1 is defined as E, E^2 as $E[E]$, and E^{n+1} as $E^n[E]$. The *infinite* unfolding represents the simplest and most intuitive solution to the equation. In the CCS grammar, such a solution is obtained by turning the equation into a constant definition, namely the constant K_E with $K_E \triangleq E[K_E]$. We call K_E the *syntactic solution of the equation*.

For a system of equations $\widetilde{X} = \widetilde{E}[\widetilde{X}]$, the unfoldings are defined accordingly (where E_i replaces X_i in the unfolding): we write $\widetilde{E}^2$ for the system $\{X_i = E_i[\widetilde{E}]\}_{i \in I}$, and similarly for $\widetilde{E}^n$. The syntactic solutions are defined to be the set of mutually recursive constants $\{K_{\widetilde{E},i} \triangleq E_i[\widetilde{K}_{\widetilde{E}}]\}_i$.

As equation expressions are terms of an extended CCS grammar, that includes variables, we can apply the SOS rules of CCS to them (assuming that variables have no transitions). We extend accordingly to expressions notations and terminology for LTS and transitions.

We have the following properties for transitions of processes of the form $E[\widetilde{P}]$, where the transition emanates from the E component only:

Lemma 1.1.3 (Expression transitions)**.**

1. *Given E and E' two equation expressions, if $E \xrightarrow{\mu} E'$, then $E[\widetilde{P}] \xrightarrow{\mu} E'[\widetilde{P}]$, for all processes $\widetilde{P}$, and $E[\widetilde{F}] \xrightarrow{\mu} E'[\widetilde{F}]$ for all equation expressions $\widetilde{F}$.*

2. *If E is a guarded expression and $E[\widetilde{P}] \xrightarrow{\mu} T$, then there is an expression E' such that $E \xrightarrow{\mu} E'$ and $T = E'[\widetilde{P}]$. Similarly for a transition $E[\widetilde{F}] \xrightarrow{\mu} E'$.*

Proof. 1. By a simple induction on the derivation of the transition.

2. By a simple induction on the equation expression E.

□

In the hypothesis of case 1 above, we sometimes call a transition $E[\widetilde{P}] \xrightarrow{\mu} E'[\widetilde{P}]$ *an instance* of the expression transition $E \xrightarrow{\mu} E'$.

Definition 1.1.4 (Reducts)**.** 1. The *set of reducts* of an expression E, written $\mathtt{red}(E)$, is given by:

$$\mathtt{red}(E) \quad \stackrel{\triangle}{=} \quad \bigcup_n \{E_n \mid E \xrightarrow{\mu_1} E_1 \cdots \xrightarrow{\mu_n} E_n \text{ for some } \mu_i, E_i \ (1 \leq i \leq n)\}.$$

2. The set of *reducts of the unfoldings* of a system of equations $\{X_i = E_i[\widetilde{X}]\}_{i \in I}$, also written $\mathtt{red}_\omega(\widetilde{E})$, is defined as

$$\mathtt{red}_\omega(\widetilde{E}) \quad \stackrel{\triangle}{=} \quad \bigcup_{n \in \mathbb{N}, i \in I} \mathtt{red}(E_i^n)$$

Definition 1.1.5. A system of equations $\widetilde{E}$ *protects its solutions* if, for all solution $\widetilde{P}$ of the equation $\widetilde{X} = \widetilde{E}[\widetilde{X}]$, the following holds: for all $E' \in \mathtt{red}_\omega(\widetilde{E})$, if $E'[\widetilde{P}] \stackrel{\mu}{\Rightarrow} Q$ for some μ and Q, then there exists E'' and n such that $E'[\widetilde{E}^n] \stackrel{\hat{\mu}}{\Rightarrow} E''$, and $E''[\widetilde{P}] \approx Q$.

Consider a single equation $X = E[X]$: it protects its solutions when all sequences of transitions emanating from $E'[P]$, where P is a solution of E and E' is a reduct of the unfoldings of E, can be mimicked by transitions involving unfoldings of E and reducts of E only (without P performing a transition). This is a technical condition, which is useful in the proofs, as a sufficient condition for unique solution. Indeed, when we give examples of equations having a unique solution, they also satisfy this property.

Proposition 1.1.6. *A system of equations that protects its solutions has a unique solution for $\approx$.*

Proof. Given two solutions $\widetilde{P}$, $\widetilde{Q}$ of the system of equations $\widetilde{X} = \widetilde{E}[\widetilde{X}]$ we prove that the relation

$$\mathcal{R} \quad \stackrel{\triangle}{=} \quad \{(S,T) \mid \exists E', \text{ s.t. } S \approx E'[\widetilde{P}],\, T \approx E'[\widetilde{Q}] \text{ and } E' \in \mathtt{red}_\omega(\widetilde{E})\}$$

is a bisimulation relation such that $\widetilde{P}\mathcal{R}\widetilde{Q}$.

We consider $(S,T) \in \mathcal{R}$, that is, $S \approx E'[\widetilde{P}]$ and $T \approx E'[\widetilde{Q}]$, for some $E' \in \mathtt{red}_\omega(\widetilde{E})$. If $S \xrightarrow{\mu} S'$, then by bisimilarity $E'[\widetilde{P}] \stackrel{\hat{\mu}}{\Rightarrow} S'' \approx S'$. Since $\widetilde{E}$ protects its solutions, there are n, E'' such that $E'[\widetilde{E}^n] \stackrel{\hat{\mu}}{\Rightarrow} E''$ and $E''[\widetilde{P}] \approx S''$. We can deduce by Lemma 1.1.3(1) that $E'[\widetilde{E}^n[\widetilde{Q}]] \stackrel{\hat{\mu}}{\Rightarrow} E''[\widetilde{Q}]$. Since $\widetilde{Q}$ is a solution of $\widetilde{E}$, we have $\widetilde{Q} \approx \widetilde{E}^n[\widetilde{Q}]$. This entails, as we know by hypothesis $T \approx E'[\widetilde{Q}]$, that $T \approx E'[\widetilde{Q}] \approx E'[\widetilde{E}^n[\widetilde{Q}]]$. By bisimilarity, we deduce from $E'[\widetilde{E}^n[\widetilde{Q}]] \stackrel{\hat{\mu}}{\Rightarrow} E''[\widetilde{Q}]$ that $T \stackrel{\hat{\mu}}{\Rightarrow} T' \approx E''[\widetilde{Q}]$.

The situation can be depicted on the following diagram:

$$\begin{array}{ccccccccccc}
S & \approx & E'[\widetilde{P}] & \approx & E'[\widetilde{E}^n[\widetilde{P}]] & & E'[\widetilde{E}^n[\widetilde{Q}]] & \approx & E'[\widetilde{Q}] & \approx & T \\
{\scriptstyle\mu}\downarrow & & {\scriptstyle\hat{\mu}}\Downarrow & & {\scriptstyle\hat{\mu}}\Downarrow & = & {\scriptstyle\hat{\mu}}\Downarrow & & & & {\scriptstyle\hat{\mu}}\Downarrow \\
S' & \approx & S'' & \approx & E''[\widetilde{P}] & & E''[\widetilde{Q}] & & \approx & & T'
\end{array}$$

Then, from $E' \in \mathtt{red}_\omega(\widetilde{E})$ and $E'[\widetilde{E}^n] \stackrel{\hat{\mu}}{\Rightarrow} E''$, we deduce that $E'' \in \mathtt{red}_\omega(\widetilde{E})$. Finally, $T' \approx E''[\widetilde{Q}]$ and $S' \approx E''[\widetilde{P}]$ give that $(S,T) \in \mathcal{R}$.

We reason symmetrically for $T \xrightarrow{\mu} T'$. □

We say that the syntactic solutions of the system $\widetilde{E}$ do not diverge if, for all $i \in I$, $K_{\widetilde{E},i}$ does not diverge.

Theorem 1.1.7 (Unique solution)**.** *A guarded system of equations whose syntactic solutions do not diverge has a unique solution for* $\approx$*.*

Proof. To enhance readability, we first give the proof for a single equation $X = E[X]$. We discuss the generalisation to a system of equations at the end of the proof.

Given a guarded equation expression E, we prove that E protects the solutions of its equations. To prove that, we need to consider a transition $E_0[P] \stackrel{\mu}{\Rightarrow} P'$, for some $E_0 \in \mathtt{red}_\omega(E)$ and some solution P of E.

We build a sequence of expressions E_n, and an increasing sequence of transitions $E_0[E^n] \stackrel{\mu?}{\Rightarrow} E_n$ (where $\stackrel{\mu?}{\Rightarrow} \stackrel{\text{def}}{=} \Rightarrow \cup \stackrel{\hat{\mu}}{\Rightarrow}$) such that: either this construction stops, yielding a transition $E_0[E^n] \stackrel{\mu}{\Rightarrow} E_n$, or the construction is infinite, therefore giving a divergence of K_E.

We build this sequence so that it additionally satisfies:

- Either we have $E_0[E^n] \stackrel{\hat{\mu}}{\Rightarrow} E_n$ and $E_n[P] \Rightarrow\!\approx P'$, or $E_0[E^n] \Rightarrow E_n$ and $E_n[P] \stackrel{\hat{\mu}}{\Rightarrow}\!\approx P'$.
- The sequence is strictly increasing: the sequence of transitions $E_0[E^{n+1}] \stackrel{\mu?}{\Longrightarrow} E_n[E]$ is a strict prefix of the sequence of transitions $E_0[E^{n+1}] \stackrel{\mu?}{\Longrightarrow} E_{n+1}$

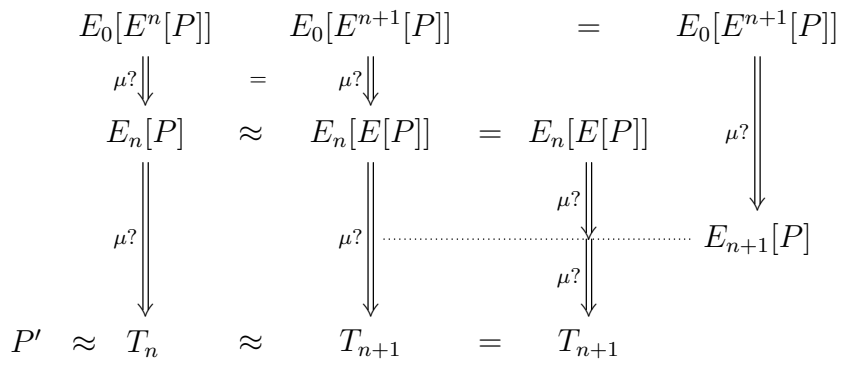


Figure 1.1: Recursion: construction of the sequence of transitions $E_0[E^n[P]] \stackrel{\mu?}{\Longrightarrow} E_n$

This construction is illustrated by Figure 1.2. We start with the empty sequence from E_0.

Suppose therefore that at step n, we have for example $E_0[E^n[P]] \stackrel{\mu}{\Rightarrow} E_n[P] \Rightarrow T_n$, with $P' \approx T_n$.

- If $E_n[P] \Rightarrow T_n$ is the empty sequence, we stop. We have in this case $E_0[E]^n \stackrel{\mu}{\Rightarrow} E_n$ and $P' \approx E_n[P]$.

- Otherwise (as depicted on Figure 1.1), we unfold further equation E: we have $E_0[E^{n+1}] \stackrel{\mu}{\Rightarrow} E_n[E]$ by Lemma 1.1.3. By congruence and bisimilarity we have $E_n[E[P]] \Rightarrow T_{n+1} \approx P'$ for some T_{n+1}. If $E_n[E[P]] \Rightarrow T_{n+1}$ is the empty sequence, we stop as previously. Otherwise, we take $E_n[E] \stackrel{\mu?}{\Longrightarrow} E_{n+1}$ to be the longest prefix sequence of transitions in $E_n[E[P]] \Rightarrow T_{n+1}$ that are instances of expression transitions from $E_n[E]$ (we remark that as $E_n[E]$ is guarded, this sequence is not empty).

Suppose now that the construction given above never stops. We know that $E_n[E] \stackrel{\mu?}{\Longrightarrow} E_{n+1}$, therefore $E_n[K_E] \stackrel{\mu?}{\Longrightarrow} E_{n+1}[K_E]$. This gives an infinite sequence of transitions starting from $E_0[K_E]$: $E_0[K_E] \stackrel{\mu?}{\Longrightarrow} E_1[K_E] \stackrel{\mu?}{\Longrightarrow} \ldots$. We observe that in the latter sequence, every step involves at least one transition, and moreover, there is at most one visible action (μ) occurring in this infinite sequence. Therefore $E_0[K_E]$ is divergent, which contradicts the hypothesis of the theorem. Hence, the construction does stop, and this concludes the proof.

Systems of equations. To extend the previous proof to systems of equations, we consider solutions $\widetilde{P}$ to the system $\widetilde{E}$, and use unfoldings of the system of equations (instead of unfoldings of a single equation). We also reason about an initial transition from $E_0[\widetilde{P}]$, where E_0 is a reduct of the unfoldings of one of the equations (i.e., $E_0 \in \mathtt{red}_\omega(\widetilde{E})$). $\square$

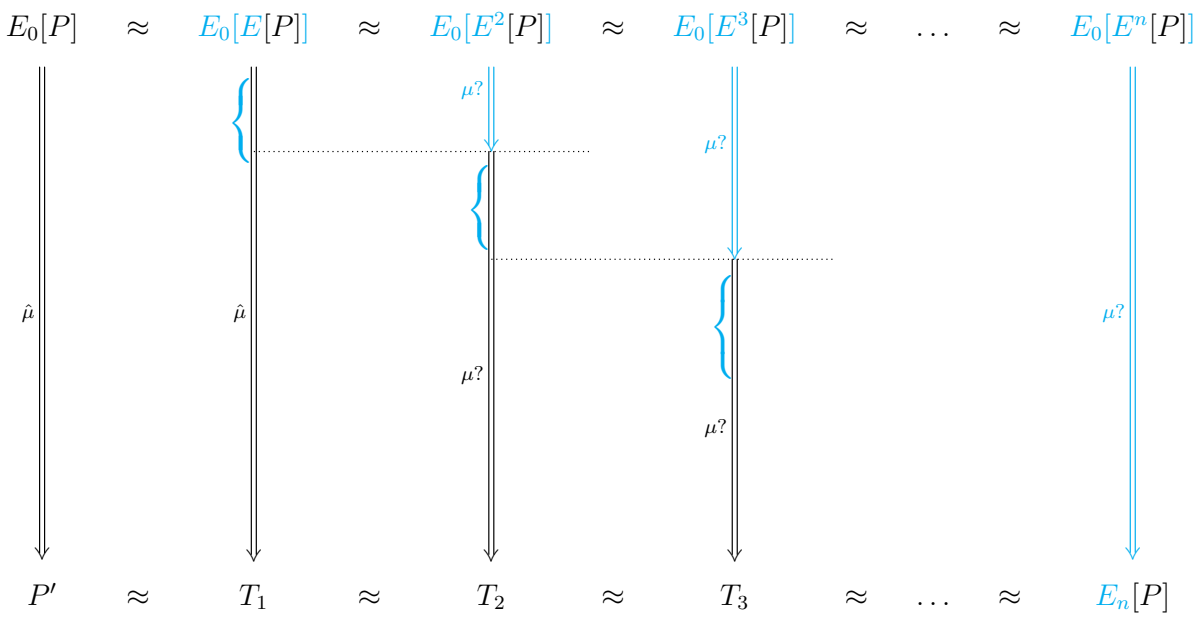


Figure 1.2: Construction of the expression transition

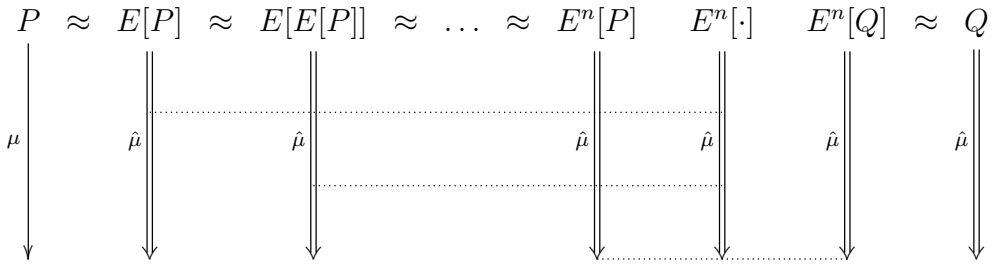


Figure 1.3: Proof of Theorem 1.1.7, building a a transition of $E^n[Q]$ where Q does not move

1.1.2 Innocuous Divergences

In this section we refine Theorem 1.1.7 by taking into account only certain forms of divergence. To introduce the idea, consider the equation $X = !\tau \mid a.X$: the divergences induced by $!\tau$ do not prevent uniqueness of the solution, as any solution P necessarily satisfies $P \approx a.P$. Indeed the variable of the equation is strongly guarded and a visible action has to be produced before accessing the variable. These divergences are not dangerous because they do not percolate through the infinite unfolding of the equation; in other words, a finite unfolding may produce the same divergence, therefore it is not necessary to go to the infinite unfolding to diverge. We call such divergences *innocuous*. Formally, these divergences are derived by applying only a finite number of times rule `const` of the LTS (see Figure 1) to the constant that represents the syntactic solution of the equation. Those divergences can be understood as instances, in the syntactic solution, of divergences of some finite unfolding

of the equation: Consider a (single) equation $X = E[X]$; this way, any divergence of E^n can be transformed into an innocuous divergence of K_E. This idea is also behind Lemma 1.2.20 below.

This refinement fits well the operational approach we adopt to formalise the results; it looks less natural in a denotational or trace-based setting for CSP like in [Ros92, Ros97], where any divergence causes a process to be considered as having an undefined behaviour (in other words, any arbitrary behaviour).

Definition 1.1.8 (Innocuous divergence)**.** Consider a guarded system of equations $\widetilde{X} = \widetilde{E}$ and its syntactic solutions $\widetilde{K}_{\widetilde{E}}$. A divergence of $K_{\widetilde{E},i}$ (for some i) is called *innocuous* when, summing up all usages of rule `const` with one of the $K_{\widetilde{E},j}$s (including $j = i$) *in all derivation proofs of the transitions belonging to the divergence*, we obtain a finite number.

Theorem 1.1.9 (Unique solution with innocuous divergences)**.** *Let $\widetilde{X} = \widetilde{E}$ be a system of guarded equations, and $\widetilde{K}_{\widetilde{E}}$ be its syntactic solutions. If all divergences of any $K_{\widetilde{E},i}$ are innocuous, then $\widetilde{E}$ has a unique solution for $\approx$.*

Proof. We reason like in the proof of Theorem 1.1.7. We explain the difference, in the case where we have a single equation (bearing in mind that the generalisation discussed at the end of the proof of Theorem 1.1.7 can be carried over accordingly).

Along the construction in that proof, if at some point the transition $E_0[E^n[P]] \stackrel{\hat{\mu}}{\Rightarrow} T_n$ is an instance of an expression transition $E_0[E^n] \stackrel{\hat{\mu}}{\Rightarrow} E_n$, then the construction can stop.

Consequently, if the construction never stops, we build a non innocuous divergence: we can assume that for any n, $E_0[E^n[P]] \stackrel{\hat{\mu}}{\Rightarrow} T_n$ is not an instance of an expression transition from $E_0[E^n]$. This means that in this sequence of transitions, the LTS rule `const` is used at least n times applied to the constant K_E. Hence, the divergence we build uses at least n times the rule `const` applied to the constant K_E, for all n. Therefore, the divergence is not innocuous, which is a contradiction. □

Remark 1.1.10. The conditions for unique solution in Theorems 1.1.7 and 1.1.9 combine syntactic (guardedness) and semantic (divergence-free) conditions. A purely semantic condition can be used if rule `const` of Figure 1 is modified so that the unfolding of a constant yields a τ-transition:

$$\frac{}{K \stackrel{\tau}{\longrightarrow} P} \text{ if } K \stackrel{\triangle}{=} P$$

Thus in the theorems the condition imposing guardedness of the equations could be dropped. The resulting theorems would actually be more powerful because they would accept equations which are not syntactically guarded: it is sufficient that each equation has a finite unfolding which is guarded. For instance the system of equations $X = b \mid Y$, $Y = a.X$ would be accepted, although the first equation is not guarded.

Remark 1.1.11. Even taking innocuous divergences into account, our criterion for equations to have a unique solution is not complete. Indeed, the following equation

$$X = a.X + \overline{a}.X + d.(X \mid X) \tag{1.1}$$

has a non-innocuous divergence: its syntactic solution, K, has the transition $K \xrightarrow{d} K \mid K$; then, $K \mid K$ diverges by unfolding on both sides at every step. However, the equation has a unique solution, intuitively because the prefix d acts as a strong guard, that does not take part in the divergence.

Note moreover that this equation is non-linear, in the sense that there are occurrences of the same variable X in parallel. In practice, such examples are not very useful. Additionally, the equation is not at top-level, meaning the syntactic solution produces some observable action before the divergence can be fired up. Completeness of Theorem 1.1.9 in respect to both linear equations and equations with top-level divergences is established for a subcalculi of CCS in Section 1.1.5.

1.1.3 Toolbox

In this section we give tools, in the form of two lemmas, to facilitate the use of Theorems 1.1.7 and 1.1.9 for concrete proofs. Both are sufficient conditions for unique solution: the first is a syntactic condition that guarantees that an equation only has innocuous divergences; the second, a way to substitute a system of equations that has a unique solution, but to which we are not able to apply the theorems, for another.

1.1.3.1 Syntactic condition for unique solution

The following lemma states a condition to ensure that all divergences produced by a system of equations are innocuous. This condition is decidable, but weak. However, in practice, it is often satisfied; it is in particular sufficient for the examples we give in CCS. With this criterion, it is also possible to forego guardedness, and replace it with a weaker hypothesis: equations only have to be *eventually guarded*, meaning, for each equation E_i, there has to be some n_i so that $E_i^{n_i}$ is guarded.

Lemma 1.1.12. *Consider a system of equations $\widetilde{X} = \widetilde{E}$, and suppose that for each i there is n_i such that in $E_i^{n_i}$, each variable is underneath a visible prefix (say, a or $\overline{a}$) whose complementary prefix ($\overline{a}$ or a) does not appear in any equation. Then the system has only innocuous divergences.*

Proof. The condition ensures that at least one of the prefix occurring above the equation variables may never take part in a τ action. This entails that this prefix cannot be triggered along an infinite sequence of τ steps. Hence a divergence of the unique solution of such a system of equations may not require the rule `const` to be used an infinite number of times. □

1.1.3.2 Extension of a system of equations

We show how it is possible to transfer the property of uniqueness of solutions from a system of equations to another one. This could be useful in the case that the first system of equations, albeit having a unique solution, has non-innocuous divergences. We mostly use it in cases

where neither equations have non-innocuous divergences, but it is harder to show the absence of divergences for one system rather than for the other. For instance, we can apply conditions similar to Lemma 1.1.12 to the second system, but not to the first.

Both system of equations need not be of the same size: on system needs to *extend* the other. Intuitively, this means Take, for instance, the system of equations

$$X = a.\, Y$$
$$Y = \tau.\, X$$

This system of equations can be turned into the equation $X = a.\, \tau.\, X$, for which it is easiest to show the absence of divergences, by using Lemma 1.1.12 for instance.

Definition 1.1.13. A system of equations $\mathcal{E}'$ *extends* a system $\mathcal{E}$ if there exists a fixed set of indices J such that any solution of $\mathcal{E}$ can be obtained from a solution of $\mathcal{E}'$ by removing the components corresponding to indices in J.

Lemma 1.1.14. *Consider two systems of equations $\mathcal{E}'$ and $\mathcal{E}$ where $\mathcal{E}'$ extends $\mathcal{E}$. If $\mathcal{E}'$ has a unique solution, then the property also holds for $\mathcal{E}$.*

Remark 1.1.15. We cannot derive Lemma 1.1.14 by comparing the syntactic solutions of the two systems $\mathcal{E}'$ and $\mathcal{E}$. For instance, the syntactic solutions of the equations $X = \tau.\, X$ and $X = \tau.\, \tau.\, \tau \ldots$ are (strongly) bisimilar; yet only the latter equation has the unique-solution property. (Further, Lemma 1.1.14 allows us to compare systems of different size.)

We use Lemma 1.1.14 in Chapter 2, when studying the system $\mathcal{E}_{\mathcal{R}}$ for the encoding of the call-by-value λ-calculus in the Internal π-calculus. For this, we show how $\mathcal{E}_{\mathcal{R}}$ can be extended into a system for which uniqueness is easier to establish.

System $\mathcal{E}_{\mathcal{R}}$ contains equations which give rise to *innocuous divergences*, therefore we could also use a version of Theorem 1.1.9; however Lemma 1.1.14 also allows us to work with two systems of equations of different sizes.

Another development of the theory in [DHS19] is presented in Section 2.2.9. It is a generalisation of Theorem 1.4.11 and Lemma 1.1.14 to trace equivalence and to trace preorders. The extension to preorders follows the line of the one presented in [DHS19].

Remark 1.1.16. Lemma 1.1.14 does not replace Theorem 1.1.9 and its treatment of innocuous divergences. Indeed, consider the equation $X = (a.\, X) \mid K$, where K is the constant defined by the recursive constant $K \stackrel{\triangle}{=} \tau.\, (K \mid b) + c$. Previous equation does have a unique solution, which can be demonstrated using Theorem 1.1.9 (it only has innocuous divergences). However, the divergence in K, albeit innocuous in this specific equation, is observable, in the sense that if $K \xrightarrow{\tau} P_0 \xrightarrow{\tau} P_1 \ldots$, the P_is are pairwise distinct for $\approx$. Therefore, any solution to the equation has a divergence. This makes Lemma 1.1.14 insufficient to treat this example.

1.1.4 Comparison with other techniques

1.1.4.1 An example (lazy and eager servers), and comparison with contractions

We now show an example of application of our technique, taken from [San15]. The example also illustrates the relative strengths of the two unique solution theorems (Theorems 1.1.7 and 1.1.9), and a few aspects of the comparison with other bisimulation techniques.

For the sake of readability, we use a version of CCS with value passing; this could be translated into pure CCS [Mil89]. In a value-passing calculus, $a(x).\,P$ is an input at a in which x is the placeholder for the value received, whereas $\overline{a}\langle n\rangle.\,P$ is an output at a of the value n; and $A\langle n\rangle$ is a parametrised constant. This example consists of two implementations of a server; this server, when interrogated by clients at a channel c, should start a certain interaction protocol with the client, after consulting an auxiliary server A at a.

We consider the two following implementations of this server: the first one, L, is 'lazy', and consults A only *after* a request from a client has been received. In contrast, the other one, E, is 'eager', and consults A *beforehand*, so then to be ready in answering a client:

$$\begin{array}{rcl} L & \triangleq & c(z).\,a(x).\,(L \mid R\langle x,z\rangle) \\ E & \triangleq & a(x).\,c(z).\,(E \mid R\langle x,z\rangle) \end{array} \qquad\qquad A\langle n\rangle \triangleq \overline{a}\langle n\rangle.\,A\langle n+1\rangle$$

Here $R\langle x,z\rangle$ represents the interaction protocol that is started with a client, and can be any process. It may use the values x and z (obtained from the client and the auxiliary server A); the interactions produced may indeed depend on the values x and z. We assume for now that $R\langle x,z\rangle$ may not use channel c and a; that is, the interaction protocol that has been spawned need not come back to the main server or to the auxiliary server. Moreover we assume R may not diverge. We want to prove that the two servers, when composed with A, yield bisimilar processes. We thus define $LS\langle n\rangle \triangleq \boldsymbol{\nu} a\,(A\langle n\rangle \mid L)$ and $ES\langle n\rangle \triangleq \boldsymbol{\nu} a\,(A\langle n\rangle \mid E)$. A proof that $LS\langle n\rangle \approx ES\langle n\rangle$ using the plain bisimulation proof method would be long and tedious, due to the differences between the lazy and the eager server, and to the fact that R is nearly an arbitrary process.

The paper [San15] presents two proofs of this equivalence. One proof makes use of the 'bisimulation up-to expansion and context' technique; this makes it possible to carry out a proof using a single pair of processes for each integer n. A proof of similar size uses the technique of 'unique solution of contractions', by establishing, with the help of a few simple algebraic laws, that $\{LS\langle n\rangle\}_n$ and $\{ES\langle n\rangle\}_n$ are solutions of the same system of contractions.

We can also build a proof using the technique of 'unique solution of equations'. Milner's original version of this technique cannot be used, because the equations make use of operators other than just prefix and sum. In contrast, Theorem 1.1.7 can be applied. The equations are: $\{X_n = c(z).\,(X_{n+1} \mid R\langle n,z\rangle)\}_n$. The proofs that the two servers are solutions can be carried out using a few algebraic laws: expansion law, structural laws for parallel composition and restriction, one τ-law. It is essentially the same proof as that for unique solution of contractions.

To apply Theorem 1.1.7, we also need to check that the equations do not produce divergences. This check is straightforward, as no silent move may be produced by interactions

along c, and any two internal communications at a are separated by a visible input at c. Moreover, by assumption, the protocol R does not produce internal divergences.

If on the other hand the hypothesis that R may not diverge is lifted, then Theorem 1.1.7 is not applicable anymore, and divergences are possible. However, such divergences are innocuous: the equation need not be unfolded an infinite number of times for the divergence to occur. We can therefore still prove the result, by appealing to the more powerful Theorem 1.1.9.

We can relax the definition of R even further and allow calls back to the main server from R itself. In this case, interactions between R and the main server (eager or lazy) may yield divergences (for instance setting $R \stackrel{\text{def}}{=} \bar{c}$). Such divergences need not be innocuous, as intuitively they require infinitely many unfolding of the body of an equation. Thus now even Theorem 1.1.9 is not applicable. (More precisely, for Theorem 1.1.9 to fail $R\langle n, z\rangle$, for each n and z, should have the possibility of performing an output at c as first visible transition.)

This becomes therefore an example in which the 'unique solution of contraction' technique is more powerful, as such technique does not rely on conditions about divergence and is therefore applicable.

Comparison with up-to techniques Milner's syntactic condition for unique solution of equations essentially allows only equations in which variables are underneath prefixes and sums. The technique is not complete [San15]; for instance it cannot handle the server example of Section 1.1.4.1.

The technique of 'unique solution of contractions' [San15] relies on the theory of an auxiliary preorder (contraction), needed to establish the meaning of 'solution'; and the soundness theorems in [San15] use a purely syntactic condition (guarded variables). In contrast, our techniques with equations do not rely on auxiliary relations and their theory, but the soundness theorems use a semantic condition (divergence) , see also Remark 1.1.10).

The two techniques are incomparable. Considering the server example of Section 1.1.4.1, the contraction technique is capable of handling also the case in which the protocol R is freely allowed to make calls back to the main server, including the possibility that, in doing this, infinitely many copies of R are spawned. This possibility is disallowed for us, as it would correspond to a non-innocuous divergence. On the other hand, when using contraction, a solution is evaluated with respect to the contraction preorder, that conveys an idea of efficiency (measured against the number of silent transitions performed). Thus, while two bisimilar processes are solutions of exactly the same set of equations, they need not be solutions of the same contractions. For instance, we can use our techniques to prove that processes $K \stackrel{\triangle}{=} \tau.\, a.\, a.\, K$ and $H \stackrel{\triangle}{=} a.\, H$ are bisimilar because solutions of the equation $X = a.\, X$; in contrast, only H is a solution of the corresponding contraction.

1.1.4.2 Completeness for up-to-context techniques

We compare our unique-solution techniques with one of the most powerful forms of enhancement of the bisimulation proof method, namely Pous 'up to transitivity, bisimilarity and

context' technique [Pou08]. This technique allows one to use 'up to weak bisimilarity', 'up to transitivity', and 'up to context' techniques together. While 'up to weak bisimilarity' and 'up to transitivity' are known to be unsound techniques [PS11], here they are combined safely thanks to a 'control relation', written below $\succ$, which satisfies a termination hypothesis. This control relation is used to make sure there is no cyclic or infinite sequence of τ transitions used to hide other potential visible transitions. In that regard, this condition is very similar to the non-divergence hypothesis of Theorem 1.1.9, as can be seen in the proofs below.

In this section, we will switch freely between equation expressions and contexts, keeping in mind that contexts are needed to study up-to techniques while equation expressions are needed for unique solution of equations.

$\overline{\mathcal{R}}$ stands for $(\approx \cup \mathcal{C}(\mathcal{R}))$, and $\mathcal{R}^+$ for the transitive closure of $\mathcal{R}$.

Definition 1.1.17. Let $\succ$ be a relation that is transitive, closed under contexts, and such that $\succ$ $(\xrightarrow{\tau}^+)$ terminates. A relation $\mathcal{R}$ is a *bisimulation up to* $\succ$ *and context* if, whenever $P \mathcal{R} Q$:

1. if $P \xrightarrow{\mu} P'$ then $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ for some Q' with $P' \ (\succ \cap \overline{\mathcal{R}})^+ \ \mathcal{C}(\mathcal{R}) \ \approx \ Q'$;

2. the converse on the transitions from Q.

We refer to [Pou08] for more details on this up-to technique, and for the the proof of its soundness.

We remark that in [Pou08], instead of the transitive closure $(\succ \cap \overline{\mathcal{R}})^+$ in Definition 1.1.17, we have a *reflexive* and transitive closure. We do not know if relaxing this technical condition breaks Theorem 1.1.18 below.

We introduce some notations in order to state the correspondence between the technique introduced in Definition 1.1.17 and systems of equations.

If $\mathcal{R}$ is a relation, then we can also view $\mathcal{R}$ as an ordered sequence of pairs (e.g., assuming some lexicographical ordering). Then $\mathcal{R}_i$ indicates the tuple obtained by projecting the pairs in $\mathcal{R}$ on the i-th component $(i = 1, 2)$.

In the following statement, the *size* of a relation is the number of pairs it contains, and the size of a system of equations is the number of equations it consists of.

Theorem 1.1.18 (Completeness with respect to up-to techniques)**.** *Suppose $\mathcal{R}$ is a bisimulation up to $\succ$ and context. Then there exists a guarded system of equations, with only innocuous divergences, that admits $\mathcal{R}_1$ and $\mathcal{R}_2$ as solutions. Moreover, this system has the same size as $\mathcal{R}$.*

Proof. Suppose $\mathcal{R} = \{(P_i, Q_i)\}_{i \in I}$ is a bisimulation up to $\succ$ and context.

We index the transitions of P_i from 1 to n_i, and write $P_i \xrightarrow{\mu_{i,j}} P'_{i,j}$ for $1 \leq j \leq n_i$.

Then, by Definition 1.1.17, for $1 \leq j \leq n_i$, there exist $C_{i,j}$ and $Q'_{i,j}$ such that we have the following diagram:

$$\begin{array}{ccccc} P_i & & \mathcal{R} & & Q_i \\ {\scriptstyle \mu_{i,j}}\downarrow & & & & \Downarrow{\scriptstyle \hat{\mu}_{i,j}} \\ P'_{i,j} & (\succ \cap (\approx \cup \mathcal{C}(\mathcal{R})))^+ \; C_{i,j}[\widetilde{P}] & \mathcal{C}(\mathcal{R}) & C_{i,j}[\widetilde{Q}] \approx & Q'_{i,j} \end{array}$$

We define the system of equations $\widetilde{X} = \widetilde{E}$ by setting $\forall i \in I,\ E_i = \sum_{j=1}^{n_i} \mu_{i,j}.\, C_{i,j}[\widetilde{X}]$. By construction, this system and $\mathcal{R}$ have the same size.

We prove that $\widetilde{P}$ is a solution of $\widetilde{X} = \widetilde{E}$. We have that $P_i \sim \sum_{j=1}^{n_i} \mu_{i,j}.\, P'_{i,j}$. Moreover, by the results in [Pou08], since $\succ (\xrightarrow{\tau}{}^+)$ terminates, $\mathcal{R} \subseteq \approx$, which implies $(\succ \cap (\approx \cup \mathcal{C}(\mathcal{R})))^+ \subseteq \approx$. Therefore, $P'_{i,j} \approx C_{i,j}[\widetilde{P}]$, and $P_i \approx \sum_{j=1}^{n_i} \mu_{i,j}.\, C_{i,j}[\widetilde{P}]$. This shows that $\widetilde{P}$ is a solution of $\widetilde{X} = \widetilde{E}$.

Since $\mathcal{R} \subseteq \approx$, we have $\widetilde{P} \approx \widetilde{Q}$, hence $\widetilde{Q}$ is also a solution of the system of equations.

It is left to prove that if for some i, $K_{\widetilde{E},i}$ has a non-innocuous divergence, then $\succ (\xrightarrow{\tau}{}^+)$ does not terminate. As this would be contradictory, we will be able to apply Theorem 1.1.9 to finish the proof.

Assume that $K_{\widetilde{E},k} \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} F_0[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau} \xrightarrow{\tau} \ldots$, and that this divergence is not innocuous. Based on this divergence, we build a sequence of equation expressions F_n and F'_n such that:

$$F_0[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau}{}^* F'_0[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau} F_1[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau}{}^* F'_1[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau} F_2[\widetilde{K}_{\widetilde{E}}] \ldots$$

In the above divergence, we have that for all n, $F_n \xrightarrow{\tau}{}^* F'_n$ (these are expression transitions) and $F'_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau} F_{n+1}[\widetilde{K}_{\widetilde{E}}]$; moreover, the latter transition is not an instance of an expression transition from F'_n (meaning that $\widetilde{K}_{\widetilde{E}}$ contributes to the transition). We will then show that $F'_n[\widetilde{P}] \xrightarrow{\tau} \succ F_{n+1}[\widetilde{P}]$. Therefore $\succ (\xrightarrow{\tau}{}^+)$ does not terminate.

F_0 is already given. Assume then that we have F_n such that there is a non innocuous divergence from $F_n[\widetilde{K}_{\widetilde{E}}]$. This entails that there exists F'_n such that $F_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau}{}^* F'_n[\widetilde{K}_{\widetilde{E}}]$, $F'_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau}$, and the latter transition is not an instance of an expression transition from F'_n. Without loss of generality, we can assume that the transitions in $F_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau}{}^* F'_n[\widetilde{K}_{\widetilde{E}}]$ are instances of expression transitions.

The fact that transition $F'_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau}$ is not an instance of an expression transition means that at least one equation expression E_i is involved in it. For readability, we assume that this transition is of the form $F'_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau} F'_n[(K_{\widetilde{E},k})_{k<i}, Q, (K_{\widetilde{E},k})_{k>i}]$ for some k and some Q, with $K_{\widetilde{E},i} \xrightarrow{\tau} Q$ (i.e., in F'_n there is only one copy of $K_{\widetilde{E},i}$, and the transition involves only $K_{\widetilde{E},i}$). The reasoning below can be adapted to cases where the τ-transition involves a more complex synchronisation, and/or several copies of $K_{\widetilde{E},i}$.

Since $E_i = \sum_j \mu_{i,j}.\, C_{i,j}[\widetilde{X}]$, there is j such that $\tau = \mu_{i,j}$ and $Q = C_{i,j}[\widetilde{K}_{\widetilde{E}}]$. We then fix F_{n+1} to be $F'_n\{X_i/C_{i,j}[\widetilde{X}]\}$. Indeed we have $F'_n[\widetilde{K}_{\widetilde{E}}] \xrightarrow{\tau} F_{n+1}[\widetilde{K}_{\widetilde{E}}]$.

We have observed that $P_i \sim \sum_j \mu_{i,j}.\, P'_{i,j}$, and $E_i = \sum_j \mu_{i,j}.\, C_{i,j}[\widetilde{X}]$, hence any transition from E_i can be mimicked by a transition of P_i. Therefore we have that $P_i \xrightarrow{\tau} P'_{i,j}$ and $F'_n[\widetilde{P}] \xrightarrow{\tau} F'_n[(P_k)_{k<i}, P'_{i,j}, (P_k)_{k>i}]$. We have that $P'_{i,j} \succ^+ C_{i,j}[\widetilde{P}]$ by construction, hence,

since $\succ$ is transitive, $P'_{i,j} \succ C_{i,j}[\widetilde{P}]$; finally, $\succ$ is closed by contexts, hence

$$F'_n[(P_k)_{k<i}, P'_{i,j}, (P_k)_{k>i}] \succ F'_n[(P_k)_{k<i}, C_{i,j}[\widetilde{P}], (P_k)_{k>i}] = F_{n+1}[\widetilde{P}] \ .$$

The sequence of equation expressions F_n and F'_n has thus been defined in such a way that $F_n \xrightarrow{\tau}{}^* F'_n$, hence $F_n[\widetilde{P}] \xrightarrow{\tau}{}^* F'_n[\widetilde{P}]$. We also have $F'_n[\widetilde{P}] \xrightarrow{\tau} \succ F_{n+1}[\widetilde{P}]$, thus $F_n[\widetilde{P}](\xrightarrow{\tau}{}^+) \succ F_{n+1}[\widetilde{P}]$. This is an infinite sequence for $(\xrightarrow{\tau}{}^+) \succ$, hence we also have non-termination for $\succ (\xrightarrow{\tau}{}^+)$. This yields a contradiction. □

Theorem 1.1.9 is essential to establish Theorem 1.1.18; Theorem 1.1.7 would be insufficient.

Remark 1.1.19. The above proof shows how to build a system of equations starting from a pair of processes, by relying on the so-called *expansion law* [Mil89]. Using this law, one can use guarded sum to express the immediate transitions of a process. Accordingly, we can turn a process into a system of equations, each consisting of a guarded sum where the continuations of prefixes are equation variables.

This form of system of equations is sometimes called a specification in the literature. Such a specification can then be used to prove bisimilarity results, using unique solution of equations. Milner's unique solution theorem (Theorem 0.5.6) can be sufficient in such a situation, as long as τ prefixes do not appear in the specification.

1.1.5 Completeness of unique solution in CCS

As discussed in Remark 1.1.11, our unique solution theorem is not complete in regard to equations that have a unique solution. We recall Equation 1.1, that has a unique solution, but also a non-innocuous divergence:

$$X = a.\,X + \overline{a}.\,X + d.\,(X \mid X) \tag{1.1}$$

This equation has two peculiarities:

- It is non-linear (multiple copies of the same variable appear in parallel)
- It diverges after a visible action

In this section we show how, by forbidding each of these two properties separately, we get completeness for our theorems (on a subcalculus of CCS).

1.1.5.1 Guardedness and unique solution

We first discuss the guardedness hypothesis of Theorem 1.1.9: we show that guardedness is essential to obtain a unique-solution theorem, as equations that are not weakly guarded do not have a unique solution.

Proposition 1.1.20. *If E is not weakly guarded, then the equation $X = E$ does not have a unique solution for $\sim$, nor for $\approx$.*

Proof. We show, by induction on the syntax of E, that $E[P \mid !a] \sim E[P] \mid !a$ for any a fresh; thus, if P is solution, so is $P \mid !a$, and E does not have a unique solution.

- If E is the trivial equation expression (equation $X = X$): $P \mid !a \sim P \mid !a$
- If $E = \boldsymbol{\nu} b\, E'$, E' is not guarded and $\boldsymbol{\nu} b\, E'[P \mid !a] \sim \boldsymbol{\nu} b\, E'[P] \mid !a$ (by induction hypothesis, and because a is fresh)
- If $E = E_1 \mid E_2$: at least one of E_1 or E_2 is not weakly guarded;
  1. if neither is, $E[P \mid !a] = E_1[P \mid !a] \mid E_2[P \mid !a] \sim E_1[P] \mid !a \mid E_2[P] \mid !a \sim E[P] \mid !a$
  2. otherwise assume only E_1 is not; then $E_2[P \mid !a] \mid !a \sim E_2[P] \mid !a$ (a fresh), and $E_1[P \mid !a] \sim E_1[P] \mid !a$ (induction hypothesis).

  (by induction hypothesis, and with algebraic laws for $\sim$).
- Otherwise, E is guarded.

□

This result only applies to single equations; a system of equations would have to not be eventually guarded. For example, the system of equations:

$$X = Y$$
$$Y = a.\, X$$

would have a unique solution.

We also show the following lemma, useful for the next section:

Lemma 1.1.21. *If E is not weakly guarded, a is fresh in E, and $P \xrightarrow{a} Q$ then $E[P] \xrightarrow{a} E'[P, Q]$ for some E' such that $E'[X, X] = E$.*

Proof. By induction on the syntax of E. □

1.1.5.2 Unique solution and top-level divergences

We first show that equations that have 'top-level' non-innocuous divergences (the syntactic solution diverges immediately) do not have a unique solution.

Definition 1.1.22 (Top-level divergence)**.** We say that a process P has a top-level divergence if there is $(P_i)_{i \in \mathbb{N}}$ such that $P_0 = P$ and $P_i \xrightarrow{\tau} P_{i+1}$ for all $i \in \mathbb{N}$.

Likewise, we say that a divergence of an equation is at top-level if it is a divergence (i.e., a sequence $(E_i)_{i \in \mathbb{N}}$ and a sequence $(\mu_i)_{i \in \mathbb{N}}$ with $E_i[E^\infty] \xrightarrow{\mu_i} E_{i+1}[E^\infty]$ and $E_0 = E$) for which $\mu_i = \tau$ for all $i \in \mathbb{N}$.

Proposition 1.1.23. *Let E be a weakly guarded equation expression such that its syntactic solution E^∞ has a non-innocuous divergence at top-level. Then E does not have a unique solution for $\approx$.*

Proof. We show there is $P \not\approx E^\infty$ that is solution of $X = E$.

Consider a non-innocuous divergence $E^\infty \xrightarrow{\tau} E_1[E^\infty] \xrightarrow{\tau} E_2[E^\infty] \dots$. It is not a divergence of E, hence there are equation expressions transitions $E \stackrel{\tau}{\Rightarrow} E_i$, such that $E_i[E^\infty] \xrightarrow{\tau}$ and this is not an equation expression transition transition of E_i. The sequence of transitions $E \stackrel{\tau}{\Rightarrow} E'$ is not empty: E is weakly guarded. The transition $E_[E^\infty] \xrightarrow{\tau}$ is not an equation expression transition of E_i, hence X is not weakly guarded in E_i.

Fix a fresh name α. We show that, for a well-chosen constant K, the recursively defined process

$$K_{\alpha,K} \stackrel{\text{def}}{=} \tau.\, E[K_{\alpha,K}] + \alpha.\, K$$

is solution of $E = X$ for $\approx$. We show that the singleton relation $\{(K_{\alpha,K}, E[K_{\alpha,K}])\}$ progresses to identity and $\approx$ (for some well-chosen K).

We start with the challenges from $E[K_{\alpha,K}]$. E is weakly guarded, therefore all transitions from $E[K_{\alpha,K}]$ are equation expressions transitions of E; thus, a transition $E[K_{\alpha,K}] \xrightarrow{\mu} E''[K_{\alpha,K}]$ can be matched by a transition $K_{\alpha,K} = \tau.\, E[K_{\alpha,K}] + \alpha.\, K \xrightarrow{\tau} E[K_{\alpha,K}] \xrightarrow{\mu} E''[K_{\alpha,K}]$.

$$\begin{array}{ccc} K_{\alpha,K} & \mathcal{R} & E[K_{\alpha,K}] \\ {\scriptstyle\mu}\Downarrow & & {\scriptstyle\mu}\downarrow \\ E''[K_{\alpha,K}] & = & E''[K_{\alpha,K}] \end{array}$$

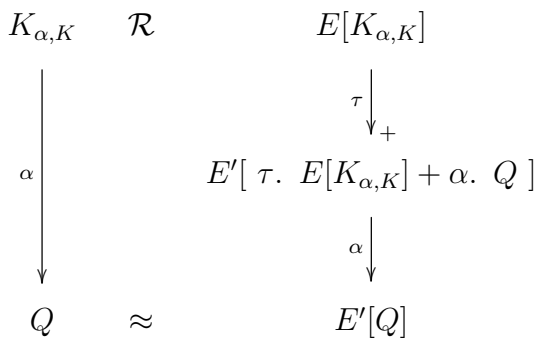
We now consider challenges from $K_{\alpha,K}$. There are 2 possible transitions:

- The transition $K_{\alpha,K} \xrightarrow{\tau} E[K_{\alpha,K}]$ is matched by the empty transition $E[K_{\alpha,K}] \stackrel{\hat{\tau}}{\Rightarrow} E[K_{\alpha,K}]$.

- The transition $K_{\alpha,K} \xrightarrow{\alpha} K$ is matched with

$$\begin{aligned} E[K_{\alpha,K}] &\stackrel{\tau}{\Rightarrow} E'[K_{\alpha,K}] \\ &= E'[\tau.\, E[K_{\alpha,K}] + \alpha.\, K] \\ &\xrightarrow{\alpha} E''[K_{\alpha,K}, K] \text{ (by Lemma 1.1.21, for some } E'') \end{aligned}$$

  K needs to be such that $K \approx E''[K_{\alpha,K}, K]$ (so that $\mathcal{R}$ progresses to $\approx$). We thus take K to be the recursively defined constant $K \stackrel{\triangle}{=} E''[K_{\alpha,K}, K]$.



We indeed have $K_{\alpha,K} \not\approx E^\infty$: $K_{\alpha,K} \xrightarrow{\alpha}$ and α is fresh in E.

Furthermore, for any fresh name β, there is K such that $K_{\beta,K}$ is solution of $E = X$. Therefore it has an infinite number of non-bisimilar solutions. □

1.1.5.3 Completeness for linear equations

The proof of the result presented in this section is cumbersome and highly combinatorial. We therefore consider it has no place in the main body of this document; we only present a proof sketch.

Definition 1.1.24. We say that an equation expression is linear if its syntax belongs to the inductively defined set:

$$E \quad \stackrel{\text{def}}{=} \quad X \ \mid\ \Sigma_{i\in I}\mu_i.\, E_i \ \mid\ \boldsymbol{\nu} a\, E \ \mid\ E \mid P \ \mid\ P \mid E$$

Additionally, the finite ν-free fragment of CCS is the subset of CCS syntax where replication, constant names, and restriction are not used.

Proposition 1.1.25. *Let E be a linear equation expression from the finite ν-free fragment of CCS. The equation $X = E$ has a unique solution for $\approx$ if and only if its syntactic solution diverges.*

The idea of the proof is to extract, from an arbitrary divergence, a 'regular' divergence, that is, a divergence where the same transitions are followed regularly, infinitely many times. In other words, we want to find a reduct E' of the equation E, such that $E'[E^n] \Rightarrow E' \mid E''$ (up to rearranging of parallel compositions) for some E'' and some n. Then E'' can be ignored, and the same route taken to create a divergence.

This is possible because equations are linear, hence they reduce to equation expressions of the shape $E_0 \mid P_1 \mid \ldots \mid P_n$, where E_0 and the P_is are subterms of E. As there are finitely many of such subterms, by a well-ordering argument there must be two points in the divergence where the same subterms are repeated (possibly with some additional ones), thus yielding a regular divergence.

From a regular divergence, one can build two solutions to the equation, by 'hiding' an fresh prefix a in the solution, at a point at which the regular divergence must have started.

1.2 Generalizations and abstract setting

1.2.1 Abstract Formulation

In this section we propose generalisations of the unique-solution theorems. For this we introduce abstract formulations of them, which are meant to highlight their essential ingredients. When instantiated to the specific case of CCS, such abstract formulations yield the theorems in Section 1.1. The proofs are adapted from those of the corresponding theorems in Section 1.1. The results of this section, up to Theorem 1.2.16, have been formalised in Coq theorem prover [Dur17], however with slight differences and only for equations with a single variable (see Remark 1.2.17).

The abstract formulation is stated on a generic LTS, that is, a triple $\mathcal{T} = (S, \Lambda, \rightarrow)$ where: S is the set of states; Λ the set of action labels, containing the special label τ accounting for silent actions; $\rightarrow$ is the transition relation. As usual, we write $s_1 \xrightarrow{\mu} s_2$ when $(s_1, \mu, s_2) \in \rightarrow$. The definition of weak bisimilarity $\approx$ is as in Section 0.2. We omit explicit reference to $\mathcal{T}$ when there is no ambiguity.

We reason about *state operations*, i.e., functions from S^n to S for some n, and use f, f', g to range over them. We say that an operation $S^n \rightarrow S$ is an operation *of arity* n. CCS processes correspond here to the states of an LTS. In turn, a CCS context C (with multiple holes) corresponds to a state operation, mapping processes $\widetilde{P}$ to the process $C[\widetilde{P}]$. If $f : S \rightarrow S$ and $f' : S \rightarrow S$ are the state operations corresponding to the unary contexts C and C', then we can indeed check that $f \circ f' : P \mapsto C[C'[P]]$ is the unary state operation corresponding to the context $C[C']$. Equations are now of the shape $X = f(X)$ (for equations with a single variable), where X is a state variable; this is in spirit similar to replace equation expressions with contexts applied to variables, e.g., equations of the shape $X = C[X]$ in CCS.

We first recall finitary multiple composition of operations.

Definition 1.2.1 (Finitary multiple composition)**.** If f is an operation $S^m \rightarrow S$ and $g_1, \ldots, g_m$ are operations $S^n \rightarrow S$, the finitary multiple composition of f and $\widetilde{g}$, written $f \circ \widetilde{g}$, is the n-ary operation

$$(x_1, \ldots, x_n) \mapsto f(g_1(x1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$$

We will usually consider composition of operations $f, \widetilde{g}$ of the same arity, meaning, if f is an operation $S^n \rightarrow S$, then so are the g_is.

Sets of functions that are closed under finitary multiple composition are studied in universal algebra [SB81]; when, additionally, such a set contains all projections $S^n \rightarrow S$ for all n, it is called a *clone*. Clones are a widely studied algebraic structure [Sze86, SB81, HP07, BP15a, BPP17], used to represent sets of finitary operations over a given set. This concept is related to that of term algebra; in particular, a term algebra can be seen as a (freely generated) clone. The image, by the function associated to a term, of a tuple of variables is simply the term itself; this function simply performs the adequate substitutions when applied to an arbitrary term.

This is similar to our approach. Therefore our use of clones in regard to systems of equations is quite mild: we simply use clones to represent sets of finitary contexts (again, contexts are seen as functions $S^n \to S$), or, equivalently, sets of finitary equation expressions.

Definition 1.2.2 (Clone of functions)**.** A set $\mathbf{C}$ of operations on a set S is a *clone of functions on* S if:

1. $\mathbf{C}$ contains all projections: $\pi^n_k : S^n \to S$, defined by $\pi^n_k(x_1, \ldots, x_n) = x_k$
2. $\mathbf{C}$ is closed under finitary multiple composition.

Given a fixed language, we usually consider congruence to be a property of an equivalence or relation, stating that the relation is preserved by applying any context to related terms (see Definition 0.2.5). Here, we rather fix the relation (for instance, bisimilarity $\approx$), and consider sets of contexts that preserve said relation. We say that such a set of contexts *respects* the relation.

Definition 1.2.3 (Respect of an equivalence)**.** A state operation $f : S^n \to S$ respects a relation $\mathcal{R}$ if, whenever $x_i \mathcal{R} y_i$ for all $i \leq n$, where $\widetilde{x}, \widetilde{y} \in S$, then $f(\widetilde{x}) \mathcal{R} f(\widetilde{y})$.
We say that a set $\mathbf{C}$ of operations respects $\mathcal{R}$ if for all $f \in \mathbf{C}$, f respects $\mathcal{R}$.

Definition 1.2.4 (Autonomy)**.** For state functions f, f' we say that there is an *autonomous μ-transition from f to f'*, written $f \xrightarrow{\mu} f'$, if for all states x it holds that $f(x) \xrightarrow{\mu} f'(x)$. Likewise, for operations f, f' of arity n, we say that there is an autonomous transition $f \xrightarrow{\mu} f'$ if for all $\widetilde{x}$, $f(\widetilde{x}) \xrightarrow{\mu} f'(\widetilde{x})$.

Likewise, given a clone $\mathbf{C}$ of state operations and $f \in \mathbf{C}$, we say that a transition $f(\widetilde{x}) \xrightarrow{\mu} y$ is *autonomous on* $\mathbf{C}$ if, for some $f' \in \mathbf{C}$ we have $f \xrightarrow{\mu} f'$ and $y = f'(\widetilde{x})$. Moreover, we say that *function f is autonomous on* $\mathbf{C}$ if all the transitions emanating from f (that is, all transitions of the form $f(\widetilde{x}) \xrightarrow{\mu} y$, for some $\widetilde{x}, y, \mu$) are autonomous on $\mathbf{C}$.

When $\mathbf{C}$ is clear, we omit it, and we simply say that a function *is autonomous*.

Thus, f is autonomous on $\mathbf{C}$ if, for some indexing set I, there are μ_i and operations $f_i \in \mathbf{C}$ such that for all $\widetilde{x}$ it holds that: $f(\widetilde{x}) \xrightarrow{\mu_i} f_i(\widetilde{x})$, for each i; the set of all transitions emanating from $f(x)$ is precisely $\cup_i \{f(\widetilde{x}) \xrightarrow{\mu_i} f_i(\widetilde{x})\}$. Autonomous transitions correspond to expression transitions in CCS, and autonomous operations correspond to guarded contexts, which do not need the contribution of their process argument to perform the first transition.

The conditions under which, intuitively, a state function behaves like a CCS context, are simple: it has to belong to a clone of operations $\mathbf{C}$ that respects a behavioural equivalence or preorder; in this section we are interested in bisimilarity, hence the relation R in the definition below should be understood to be $\approx$. Hence, such clone of operations are considered relative to a behavioural equivalence or preorder.

The autonomous transitions of a set of $\mathcal{R}$-operations yield an LTS whose states are the operations themselves. Such transitions are of the form $f \xrightarrow{\mu} g$. We define weak transitions as follows: $f \Rightarrow g$ if there is a sequence of autonomous transitions $f \xrightarrow{\tau} f_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} f_n \xrightarrow{\tau} g$. $\stackrel{\mu}{\Rightarrow}$ is then simply $\Rightarrow\xrightarrow{\mu}\Rightarrow$.

We use state functions to express equations, such as $X = f(X)$. We thus look at conditions under which such an equation has a unique solution. Thinking of functions as equation expressions, to formulate our abstract theory about unique solution of equations, we have to define the divergences of finite and infinite unfoldings of state functions. The finite unfoldings of state operations $\widetilde{f} \in \mathbf{C}$ of arity n are the operations of same arity, defined recursively as follows: $f_i^0 = \pi_i^n$, and $f_i^{k+1} = f_i^k \circ \widetilde{f}$.

We also have to reason about the infinite unfolding of an equation $\widetilde{X} = \widetilde{f}(\widetilde{X})$. For this, given a clone of functions $\mathbf{C}$ on the states S, we extend S with a set of points, defined by the inductive syntax (with a single constructor)

$$g \circ \widetilde{f}^\infty \qquad\qquad g, f_1, \ldots, f_n \in \mathbf{C} \text{ of arity n}$$

(This is a syntactic construction. Note that this syntax only deals with infinite compositions, and no such terms belongs to the original clone o operations. Since a clone $\mathbf{C}$ is closed under finite compositions, we do not need to handle finite composition in the above definition)

We write $g \circ (\widetilde{h} \circ \widetilde{f}^\infty)$ for $(g \circ \widetilde{h}) \circ \widetilde{f}^\infty$. We also write f_i^∞ for $\pi_{\mathtt{i}}^{\mathtt{n}} \circ \widetilde{f}^\infty$, and call $\widetilde{f}^\infty$ the fixpoints of $\widetilde{f} = (f_1^\infty, \ldots, f_n^\infty)$. For a single operation f, we write f^∞ for $\pi_1^1 \circ f^\infty$.

We define the autonomous transitions for such terms using the following rules:

$$\frac{g \xrightarrow{\mu} g'}{g \circ \widetilde{f}^\infty \xrightarrow{\mu} g' \circ \widetilde{f}^\infty} \qquad\qquad \frac{}{g \circ (\widetilde{f}^\infty) \xrightarrow{\tau} (g \circ \widetilde{f}) \circ \widetilde{f}^\infty}\ g \text{ not autonomous} \tag{1.2}$$

Intuitively a term is 'unfolded', with the second rule, until at least one autonomous transition is uncovered, and then transitions are computed using the first rule These rules are consistent for finite composition of functions in $\mathbf{C}$, in the sense that they allow one to infer the correct transitions for finite compositions of functions (cf. Lemma 1.2.7 and 1.2.13). We thus extend weak bisimilarity $\approx$ to these terms also.

Intuitively, an infinite term has a divergence if none of its finite unfoldings ever yields an autonomous function, as it will always be possible to unfold using the second rule. We thus defined divergences:

Definition 1.2.5 (Divergences)**.** Let $(f_i)_{i\in\mathbb{N}}$ be operations in a clone $\mathbf{C}$, and consider the LTS induced by the autonomous transitions of operations in $\mathbf{C}$. A sequence of transitions $f_1 \xrightarrow{\mu_1} f_2 \xrightarrow{\mu_2} f_3 \ldots$ is a *divergence* if for some $n \geq 1$ we have $\mu_i = \tau$ whenever $i \geq n$. We also say that f_1 *diverges*. We apply these notations and terminology also to terms $g \circ \widetilde{f}^\infty$: $f_1 \circ \widetilde{f}^\infty \xrightarrow{\mu_1} f_2 \circ \widetilde{f}^\infty \ldots$ is a divergence if for $i \geq n$, $\mu_i = \tau$.

We say that the fixpoints $\widetilde{f}^\infty$ of operations $\widetilde{f}$ diverge if there is i such that f_i^∞ diverges.

In the remainder of the section we fix a clone $\mathbf{C}$ of operations and we only consider autonomous transitions on $\mathbf{C}$. Where the underlying set $\mathbf{C}$ of operations is clear, we simply call a function belonging to $\mathbf{C}$ a *operation*.

Definition 1.2.6 (Solution of an equation)**.** We say that states $\widetilde{x} \in S^n$ are solution of the system of equations

$$\{X_i = f_i(\widetilde{X})\}_{1\leq i\leq n}$$

for some operations $\widetilde{f} \in \mathbf{C}$ of same arity n if $x_i \approx f_i(\widetilde{x})$ for all $i \leq n$.

Before carrying out the proof of unique solution, we have to relate the transitions of a fixpoint and the transitions of the finite unfoldings, more precisely turn transitions of finite unfoldings into transitions of the fixpoints (the converse is not a problem: by definition, if $g \circ \widetilde{f}^\infty \xrightarrow{\mu} h \circ \widetilde{f}^\infty$, then either $g \xrightarrow{\mu} h$ or $h = g \circ \widetilde{f}$ and $\mu = \tau$). This however is complicated by the very lax conditions on a clone of operations: for instance, consider two functions $f, g \in \mathbf{C}$, g autonomous on $\mathbf{C}$, such that $g \circ f \xrightarrow{\mu} g'$. This does not imply that there is a function $h \in \mathbf{C}$ such that $g \xrightarrow{\mu} h$ and $g' = h \circ f$; indeed, for all $x \in S$, there is $h_x \in \mathbf{C}$ such that $g \xrightarrow{\mu} h_x$ and $h_x(f(x)) = g'(x)$, but these functions might differ pointwise, and a function h such that $g' = h \circ f$ might not be in $\mathbf{C}$. Rules 1.2 enforce that no unfolding might happen after an autonomous operation is reached, so to prevent divergences; this means that a transition $f^n \xrightarrow{\mu} g$ might have no equivalent in the LTS of f^∞.

To solve this problem and establish a partial correspondence between the transitions of the f^ns and of f^∞, we only consider integers minimal so that $g \circ f^n$ is autonomous (or so that it is not autonomous). We thus define

$$\mathbf{min}_{\widetilde{f}}(g) \triangleq \{n \mid g \circ f^n \text{ is not autonomous, or } n \text{ is minimal so that it is}\}. \tag{1.3}$$

We also use a finer definition of the set $\mathtt{red}_\omega(f)$, compared to the previous section. At the end of this section, to define innocuous divergences, we discuss a complementary hypothesis thanks to which we establish a perfect correspondence (Lemma 1.2.19) – notwithstanding τ-transitions used to unfold the syntactic solution.

Lemma 1.2.7. *Let $\widetilde{f}$, g and h be operations in some set $\mathbf{C}$, and $n \in \mathbf{min}_{\widetilde{f}}(g)$. Then $g \circ f^n \xrightarrow{\mu} h$ implies that $g \circ \widetilde{f}^\infty \Rightarrow\!\!\xrightarrow{\mu} h \circ \widetilde{f}^\infty$.*

Proof. Immediate derivation from Rules 1.2. □

From there we deduce the proper definition for the reducts of operations.

Definition 1.2.8 (Reducts)**.** 1. The *set of p-step reducts* of the unfoldings of the k-ary operations $\widetilde{f}$, written p-$\mathtt{red}(\widetilde{f})$, is defined inductively by:

- 0-$\mathtt{red}(\widetilde{f}) \triangleq \{\pi_i^k \mid i \leq k\}$.
- $(p{+}1)$-$\mathtt{red}(\widetilde{f}) \triangleq \{g \mid h \circ \widetilde{f}^n \xrightarrow{\mu} g \text{ for some } h \in p\text{-}\mathtt{red}(\widetilde{f}), n \in \mathbf{min}_{\widetilde{f}}(h), \text{ and } \mu \in \Lambda\}$.

2. The set of *reducts of the unfoldings* of operations $\widetilde{f}$ of arity k, also written $\mathtt{red}_\omega(\widetilde{f})$, is defined as

$$\mathtt{red}_\omega(\widetilde{f}) \triangleq \bigcup_{n \in \mathbb{N}} n\text{-}\mathtt{red}(\widetilde{f})$$

Note that if $g \in \mathtt{red}_\omega(\widetilde{f})$, it has the same arity as any of the f_i.

From the definition we immediately get the following proposition, relating reducts of the finite unfoldings and reducts of the infinite unfoldings:

Proposition 1.2.9. *Let $\widetilde{f} \in \mathbf{C}$ be operations of arity k. If $g \in \mathtt{red}_\omega(\widetilde{f})$, then there is $\mu_1, \ldots, \mu_n$ and $g_1, \ldots, g_n \in \mathtt{red}_\omega(\widetilde{f})$, with $g_1 = \pi^k_j$ for some $j \leq k$ and $g_n = g$, such that $g_i \circ \widetilde{f}^\infty \xrightarrow{\mu_{i+1}} g_{i+1} \circ \widetilde{f}^\infty$ for $i < n$.*

Proof. By induction on the predicate $g \in p\text{-}\mathtt{red}(\widetilde{f})$ (by definition it holds for some p). If $g \in p+1\text{-}\mathtt{red}(\widetilde{f})$, then there is $h \in p\text{-}\mathtt{red}(\widetilde{f})$ and $n \in \mathbf{min}_{\widetilde{f}}(h)$ such that $h \circ \widetilde{f}^n \xrightarrow{\mu} g$; by Lemma 1.2.7, $h \circ \widetilde{f}^\infty \overset{\mu}{\Rightarrow} g$. □

Proposition 1.2.9 allows us to establish that equations that have no divergences are 'eventually guarded' or 'eventually autonomous' (as defined in Section 1.1.3.1):

Lemma 1.2.10. *If the fixpoints $\widetilde{f}^\infty$ of $\widetilde{f}$ do not diverge, then for any $g \in \mathtt{red}_\omega(\widetilde{f})$, there is p such that $g \circ \widetilde{f}^p$ is autonomous.*

Proof. By contradiction: otherwise, for any p we have $g \circ \widetilde{f}^p \circ \widetilde{f}^\infty \xrightarrow{\tau} g \circ \widetilde{f}^{p+1} \circ \widetilde{f}^\infty$. Thus $g \circ \widetilde{f}^\infty$ diverges, and by Proposition 1.2.9 $\widetilde{f}^\infty$ diverges. □

We can always choose such a p to be minimal among p's that make $g \circ \widetilde{f}^p$ autonomous. We have, in particular, that for any i, there is p such that f_i^p is autonomous.

To solve the problems of the operational correspondence between $\widetilde{f}^\infty$ and the finite unfoldings, we need one last definition.

Definition 1.2.11 (Transitions up to $\widetilde{f}$)**.** Let $h, \widetilde{f} \in \mathbf{C}$ and $g \in \mathtt{red}_\omega(\widetilde{f})$. We say there is a μ-transition up to $\widetilde{f}$ from g to h, written $g \xrightarrow{\mu}_{\widetilde{f}} h$, if there is $n \in \mathbf{min}_{\widetilde{f}}(g)$ such that $g \circ \widetilde{f}^n \xrightarrow{\mu} h$. As usual, we write $\Rightarrow_{\widetilde{f}}$ for $\xrightarrow{\tau}{}^*_{\widetilde{f}}$, $\overset{\mu}{\Rightarrow}_{\widetilde{f}}$ for $\Rightarrow_{\widetilde{f}} \xrightarrow{\mu}_{\widetilde{f}} \Rightarrow_{\widetilde{f}}$ and $\overset{\hat{\mu}}{\Rightarrow}_{\widetilde{f}}$ for either $\overset{\mu}{\Rightarrow}_{\widetilde{f}}$ or $\Rightarrow_{\widetilde{f}}$ when $\mu = \tau$.

Most importantly, if $g \overset{\mu}{\Rightarrow}_{\widetilde{f}} h$, then there is n such that $g \circ \widetilde{f}^n \overset{\mu}{\Rightarrow}$ (by definition of autonomy). It also holds that if $g \xrightarrow{\mu}_{\widetilde{f}} h$, $g \circ \widetilde{f}^\infty \xrightarrow{\mu} h \circ \widetilde{f}^\infty$ (by Lemma 1.2.7). This gives us a last lemma, to turn transitions of the finite unfoldings into transitions of the fixpoints:

Lemma 1.2.12. *Let $\widetilde{f} \in \mathbf{C}$ and $g \in \mathtt{red}_\omega(\widetilde{f})$. If $g \overset{\mu}{\Rightarrow}_{\widetilde{f}} h$, then $g \circ \widetilde{f}^\infty \overset{\mu}{\Rightarrow} h \circ \widetilde{f}^\infty$ and $h \in \mathtt{red}_\omega(\widetilde{f})$.*

Proof. Proceed by induction on the length of $\overset{\mu}{\Rightarrow}_{\widetilde{f}}$, using Lemma 1.2.7. It is immediate that $h \in \mathtt{red}_\omega(\widetilde{f})$. □

Lastly, the converse (turning transitions of the fixpoints $\widetilde{f}^\infty$ into transitions of the finite unfoldings) is very simple.

Lemma 1.2.13. *If $g \in \mathtt{red}_\omega(\widetilde{f})$ and $g \circ \widetilde{f}^\infty \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} h \circ \widetilde{f}^\infty$ for some h, n and $(\mu_i)_{i \leq n}$, then there is p such that $g \circ \widetilde{f}^p \xrightarrow{\hat{\mu}_1} \cdots \xrightarrow{\hat{\mu}_n} h$.*

Proof. We choose p to be the number of times the unfolding rule is applied in the sequence of transition $g \circ \widetilde{f}^\infty \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} h \circ \widetilde{f}^\infty$. We then proceed by induction on n. □

We now present results necessary to carry out the proof of unique solution.

The proof is essentially the same as that of Theorem 1.1.7, replacing equations expressions with operations, from a fixed set of operations $\mathbf{C}$, and replacing instantiation of equation expression transitions with instantiation of autonomous transitions. A guarded equation expression becomes an autonomous operator.

There is one difference, however, as the rule of unfolding above uses a τ-transition, similarly to the alternative rule from Remark 1.1.10. The guardedness condition, that translates here to an autonomy condition, can thus be dropped.

Definition 1.2.14. Operations $\widetilde{f} \in \mathbf{C}$ *protect their solutions* if, for all solution $\widetilde{x}$ of the equation $\widetilde{X} = \widetilde{f}(\widetilde{X})$, the following holds: for all $g \in \mathtt{red}_\omega(\widetilde{f})$, if $g(\widetilde{x}) \stackrel{\mu}{\Rightarrow} y$ for some μ and y, then there exists $h \in \mathtt{red}_\omega(\widetilde{f})$ such that $g \stackrel{\hat{\mu}}{\Rightarrow}_{\widetilde{f}} h$, and $h(\widetilde{x}) \approx y$.

Proposition 1.2.15. *Let $\mathbf{C}$ be a clone that respects $\approx$, and let $\widetilde{f} \in \mathbf{C}$ be operations of arity n that protect their solutions. Then the equation $\widetilde{X} = \widetilde{f}(\widetilde{X})$ at most one solution for $\approx$.*

Proof. Given two solutions $\widetilde{x}$, $\widetilde{y}$ of the system of equations $\widetilde{X} = \widetilde{f}(\widetilde{X})$ we prove that the relation

$$\mathcal{R} \quad \stackrel{\triangle}{=} \quad \{(x, y) \mid x \approx g(\widetilde{x}) \text{ and } y \approx g(\widetilde{y}) \text{ for some } g \in \mathtt{red}_\omega(\widetilde{f})\}$$

is a bisimulation relation such that $\widetilde{x}\, \mathcal{R}\, \widetilde{y}$.

We consider $(x, y) \in \mathcal{R}$, that is, $x \approx g(\widetilde{x})$ and $y \approx g(\widetilde{y})$, for some $g \in \mathtt{red}_\omega(\widetilde{f})$. Now consider a transition $x \stackrel{\mu}{\longrightarrow} x'$; we deduce:

- $g(\widetilde{x}) \stackrel{\hat{\mu}}{\Rightarrow} x'' \approx x'$ (g respects bisimilarity).
- $g \circ \widetilde{f}^n \stackrel{\hat{\mu}}{\Rightarrow} h$ and $h(\widetilde{x}) \approx x''$ for some n, $h \in \mathtt{red}_\omega(\widetilde{f})$ ($\widetilde{f}$ protects their solutions and Lemma 1.2.12).
- $g \circ \widetilde{f}^n(\widetilde{y}) \stackrel{\hat{\mu}}{\Rightarrow} h(\widetilde{y})$ (by definition of autonomy).
- $\widetilde{y} \approx \widetilde{f}^n(\widetilde{y})$ ($\widetilde{y}$ is solution of $\widetilde{f}$, $\widetilde{f}$ respect $\approx$).
- $y \approx g(\widetilde{y}) \approx g \circ \widetilde{f}^n(\widetilde{y})$ (g respects $\approx$).
- $y \stackrel{\hat{\mu}}{\Rightarrow} y' \approx h(\widetilde{y})$ (by bisimilarity, from $g \circ \widetilde{f}^n(\widetilde{y}) \stackrel{\hat{\mu}}{\Rightarrow} h(\widetilde{y})$).

The situation can be depicted on the following diagram:

$$\begin{array}{ccccccccccc}
S & \approx & g(\widetilde{x}) & \approx & g \circ \widetilde{f}^n(\widetilde{x}) & & g \circ \widetilde{f}^n(\widetilde{y}) & \approx & g(\widetilde{y}) & \approx & y \\
\mu\downarrow & & \hat{\mu}\Downarrow & & \hat{\mu}\Downarrow & = & \hat{\mu}\Downarrow & & & & \hat{\mu}\Downarrow \\
x' & \approx & x'' & \approx & h(\widetilde{x}) & & h(\widetilde{y}) & & \approx & & y'
\end{array}$$

We have $h \in \mathtt{red}_\omega(\widetilde{f})$; thus, $y' \approx h(\widetilde{y})$ and $x' \approx h(\widetilde{x})$ give that $(x, y) \in \mathcal{R}$.

We reason symmetrically for $y \stackrel{\mu}{\longrightarrow} y'$. □

Theorem 1.2.16 (Unique solution, abstract formulation)**.** *Let $\widetilde{f} \in \mathbf{C}$ be operations of arity n such that operations in $\mathbf{C}$ respect $\approx$, and the fixpoints $\widetilde{f}^\infty$ do not diverge. Then $\widetilde{X} = \widetilde{f}(\widetilde{X})$ has at most one solution.*

Proof. We show that $\widetilde{f}$ protects its solutions.

To prove that, we need to consider a transition $g(\widetilde{x}) \stackrel{\mu}{\Rightarrow} y$, for some $g \in \mathtt{red}_\omega(\widetilde{f})$ and some solutions $\widetilde{x}$ of $\widetilde{X} = \widetilde{f}(\widetilde{X})$.

We build a sequence of operations f_n, and an increasing sequence of transitions $g \stackrel{\mu?}{\Rightarrow}_{\widetilde{f}} f_n$ (where $\stackrel{\mu?}{\Rightarrow}_{\widetilde{f}} \stackrel{\text{def}}{=} \Rightarrow_{\widetilde{f}} \cup \stackrel{\hat{\mu}}{\Rightarrow}_{\widetilde{f}}$) such that: either this construction stops, yielding a transition $g \stackrel{\mu}{\Rightarrow}_{\widetilde{f}} f_n$, or the construction is infinite, therefore giving a divergence of $\widetilde{f}^\infty$ (by repeated application of Lemma 1.2.12).

We build this sequence so that it additionally satisfies:

- Either we have $g \stackrel{\hat{\mu}}{\Rightarrow}_{\widetilde{f}} f_n$ and $f_n(\widetilde{x}) \Rightarrow \approx y$, or $gf^n \Rightarrow_{\widetilde{f}} f_n$ and $f_n(\widetilde{x}) \stackrel{\hat{\mu}}{\Rightarrow}_{\widetilde{f}} \approx y$.
- The sequence is strictly increasing: the sequence of transitions $g \stackrel{\mu?}{\Rightarrow}_{\widetilde{f}} f_n$ is a strict prefix of the sequence of transitions $g \stackrel{\mu?}{\Rightarrow}_{\widetilde{f}} f_{n+1}$.

We initialise with the empty sequence from g.

Suppose therefore that at step n, we have for example $g \stackrel{\mu}{\Rightarrow}_{\widetilde{f}} f_n$ and $f_n(\widetilde{x}) \Rightarrow y_n$, for some $y_n \approx y$.

- If $f_n(\widetilde{x}) \Rightarrow y_n$ is the empty sequence, we stop. We have in this case $g \circ \widetilde{f}^n \stackrel{\mu}{\Rightarrow} f_n$ (see Definition 1.2.11) and $y \approx f_n(\widetilde{x})$.
- Otherwise we unfold further equation $\widetilde{f}$: let p be minimal such that $f_n \circ \widetilde{f}^p$ is autonomous (there exists such a p by Lemma 1.2.10); then $p \in \mathbf{min}_{\widetilde{f}}(f_n)$. We have $g \stackrel{\mu}{\Rightarrow}_{\widetilde{f}} f_n$, and thus, by definition of autonomy and using Definition 1.2.11, $g \circ \widetilde{f}^{p+k} \stackrel{\mu}{\Rightarrow} f_n \circ \widetilde{f}^p$ for some k. By bisimilarity and respect of bisimilarity, we have $f_n \circ \widetilde{f}^p(\widetilde{x}) \Rightarrow y_{n+1} \approx y$ for some y_{n+1}. If $f_n \circ \widetilde{f}^p(\widetilde{x}) \Rightarrow y_{n+1}$ is the empty sequence, we stop as previously. Otherwise, we take $f_n \circ \widetilde{f}^p \xrightarrow{\mu?} f_{n+1}$ to be the first transition in $f_n \circ \widetilde{f}^p(\widetilde{x}) \Rightarrow y_{n+1}$ (we remark that as $f_n \circ \widetilde{f}^p$ is autonomous, this is indeed an autonomous transition).

Suppose now that the construction given above never stops. We know that $f_n \circ \widetilde{f}^p \xrightarrow{\mu?} f_{n+1}$ for some $p \in \mathbf{min}_{\widetilde{f}}(f_n)$, therefore $f_n \xrightarrow{\mu?}_{\widetilde{f}} f_{n+1}$ and thus $f_n \circ \widetilde{f}^\infty \stackrel{\mu?}{\Longrightarrow} f_{n+1} \circ \widetilde{f}^\infty$ (by Lemma 1.2.12). This gives an infinite sequence of transitions starting from $g \circ \widetilde{f}^\infty$. In this sequence, every step involves at least one transition, and moreover, there is at most one visible action (μ) occurring in this infinite sequence. Therefore $g \circ \widetilde{f}^\infty$ is divergent, which contradicts the hypothesis of the theorem. Hence, the construction does stop, and this concludes the proof. □

The equation in the statement of the theorem might have no solution at all. For example, consider the LTS $(\mathbb{N}, \{a\}, \to)$ where for each n we have $n+1 \xrightarrow{a} n$. The operation of arity 1 f with $f(n) = n+1$ is an operation of the set $\{f^n\}_{n\in\mathbb{N}}$ (with $f^0 = \mathbf{id}$, the identity function). This set can be made into a clone: we consider its closure by composition with projections. The function f is autonomous because, for all n, the only transition of $f(n)$ is $f(n) \xrightarrow{a} n$ (this transition is autonomous because $f \xrightarrow{a} \mathtt{Id}$). A fixpoint of f would be an element x with $x \xrightarrow{a} x$, and there is no such x in the LTS.

Remark 1.2.17 (Coq Implementation [Dur17])**.** A theorem similar to Theorem 1.2.16, for an abstract formulation from [DHS19], is formalized in the proof assistant Coq [Tea], so to illustrate the flexibility of this framework, and its possible uses for mechanized proofs. There are however some differences with the version presented in this section:

1. To simplify the proof, only unary operations, (functions $S \to S$) are considered.
2. The unfolding rules of f^∞ are different, similar in spirit the constant rule in CCS (cf. Remark 1.1.10). However, in exchange, an additional hypothesis is requested. These differences are discussed in Section 1.3.1, and detailed in Appendix C.3.

This abstract approach, relying on functions, is unusual for mechanized proofs: for instance, we have to assume extensional equality of functions, which is not always harmless in this context.
However, the final proof is surprisingly simple and clean, possibly indicating that the chosen level of abstraction is appropriate for formal proofs; this abstract framework might prove well suited to formally instantiate the unique-solution proof technique to various languages (rather than, for instance, rule formats).

Theorem 1.2.16 can be refined along the lines of Theorem 1.1.9. For this, we have to relate the divergences of any $\widetilde{f}_i^n$ (for $n \geq 1$) to divergences of $\widetilde{f}^\infty$, in order to distinguish between innocuous and non-innocuous divergences. To do so, we enforce a new hypothesis, corresponding in some sense to the ability to syntactically decompose autonomous operations: the idea is that if f is autonomous, it must contain some combination of operators of the language that enforce autonomy; these operators must still appear in a context $f \circ g$, obtained by combining f with some additional syntax; thus, if $f \circ g \xrightarrow{\mu} h$, there must be some f' such that $f \xrightarrow{\mu} f'$ (this is always the case) and such that, additionally, $f' \circ g = h$. We only have to enforce this property on reducts of an operation.

Definition 1.2.18 (Decomposability.)**.** Let $\widetilde{f} \in \mathbf{C}$. We say that $\widetilde{f}$ are decomposable if whenever $g \circ \widetilde{f}^n \xrightarrow{\mu} h$ for some $h \in \mathbf{C}$ and $g \in \mathtt{red}_\omega(\widetilde{f})$ autonomous, then there is $g' \in \mathbf{C}$ such that $g \xrightarrow{\mu} g'$ and $h = g' \circ \widetilde{f}^n$.

With this hypothesis we recover an equivalence between transitions $\overset{\mu}{\Rightarrow}_{\widetilde{f}}$ and $\overset{\mu}{\Rightarrow}$.

The following lemma shows that fixpoints of a decomposable operation have the same weak transitions $\overset{\mu}{\Rightarrow}$ as their counterpart finite unfoldings. To build a transition $g \circ \widetilde{f}^\infty \overset{\mu}{\Rightarrow} g' \circ \widetilde{f}^\infty$ from a transition of an unfolding $g \circ \widetilde{f}^n \xrightarrow{\mu} h$, we need to reason up to (finite)

unfoldings of $\widetilde{f}$: thus we set $=_{\widetilde{f}}$ to be the symmetric reflexive transitive closure of the relation on operations of arity n (assuming $\widetilde{f}$ are of arity n), that relates g and g' whenever $g = g' \circ \widetilde{f}$.

Lemma 1.2.19. *If $\widetilde{f} \in \mathbf{C}$ are decomposable, then whenever $g \circ \widetilde{f}^n \stackrel{\mu}{\Rightarrow} h$, there is h' such that $h =_{\widetilde{f}} h'$, and $g \circ \widetilde{f}^\infty \stackrel{\mu}{\Rightarrow} h' \circ \widetilde{f}^\infty$.*

Proof. By induction on the length of $\stackrel{\mu}{\Rightarrow}$. For the base case, we have to show that if $g \circ \widetilde{f}^n \stackrel{\mu}{\longrightarrow} h$, then $g \circ \widetilde{f}^\infty \stackrel{\mu}{\Rightarrow} h' \circ \widetilde{f}^\infty$ for $h' =_{\widetilde{f}} h$. If $n \in \mathbf{min}_{\widetilde{f}}(g)$, then $g \circ \widetilde{f}^\infty \Rightarrow \stackrel{\mu}{\longrightarrow} h \circ \widetilde{f}^\infty$. Otherwise, take p minimal so that $g \circ \widetilde{f}^p$ is autonomous ($p > n$). We know that $g \circ \widetilde{f}^p \circ \widetilde{f}^{n-p} \stackrel{\mu}{\longrightarrow} h$; by decomposability, there is $h' =_{\widetilde{f}} h$ such that $g \circ \widetilde{f}^p \stackrel{\mu}{\longrightarrow} h'$. Thus, $g \circ \widetilde{f}^\infty \Rightarrow g \circ \widetilde{f}^p \circ \widetilde{f}^\infty \stackrel{\mu}{\longrightarrow} h' \circ \widetilde{f}^\infty$. □

Lemma 1.2.20. *Consider decomposable operations $\widetilde{f}$ in $\mathbf{C}$ and a divergence of f_i^n*

$$f_i^n \stackrel{\mu_1}{\longrightarrow} h_1 \stackrel{\mu_2}{\longrightarrow} \ldots \stackrel{\mu_k}{\longrightarrow} h_k \stackrel{\tau}{\longrightarrow} h_{k+1} \stackrel{\tau}{\longrightarrow} \ldots$$

This yields a divergence of f_i^∞: $f_i^\infty \stackrel{\mu_1}{\Longrightarrow} g_1 \circ \widetilde{f}^\infty \stackrel{\mu_2}{\Longrightarrow} \ldots \stackrel{\mu_k}{\Longrightarrow} g_k \circ \widetilde{f}^\infty \stackrel{\tau}{\Rightarrow} g_{k+1} \circ \widetilde{f}^\infty \stackrel{\tau}{\Rightarrow} \ldots$ such that for all $j \geq 1$, $g_j \in \mathtt{red}_\omega(\widetilde{f})$ and $g_j =_{\widetilde{f}} h_j$.

Given a divergence Δ of f_i^n, we write Δ^∞ to indicate the divergence of f_i^∞ obtained from Δ as in Lemma 1.2.20. We call a divergence of $\widetilde{f}^\infty$ *innocuous* when it can be described in this way, that is, as a divergence Δ^∞ of f_i^∞ obtained from a divergence Δ of f_i^n, for some n and some i.

Theorem 1.2.21 (Unique solution with innocuous divergences, abstract formulation)**.** *Let $\widetilde{f} \in \mathbf{C}$ be operations of arity n such that operations in $\mathbf{C}$ respect $\approx$. If all divergences of $\widetilde{f}^\infty$ are innocuous, then $\widetilde{X} = \widetilde{f}(\widetilde{X})$ has at most one solution.*

Proof. Just as in CCS, where the proof of Theorem 1.1.7 has to be modified for Theorem 1.1.9, here the proof of Theorem 1.2.16 is to be modified. The modification is essentially the same as in CCS, again substituting equation expressions for operations. □

1.2.2 Reasoning with other behavioural equivalences

We can adapt the results of the previous section about bisimilarity to other settings, including both preorders and non-coinductive relations. As an example, we consider trace-based relations.

The definitions for trace-based behavioural relations are given in Section 0.2.3.

The definitions from Section 1.3.1 are the same as for $\approx$; however we now consider sets of $\subseteq_{tr}$-operators, i.e., we are interested in operations that respect $\subseteq_{tr}$. Indeed the preorder $\subseteq_{tr}$ is used in the proof of unique solution for $\approx_{tr}$, hence operations must respect $\subseteq_{tr}$ rather than $\approx_{tr}$.

The notion of trace is extended to operations like we do for weak transitions: we write $f \stackrel{s}{\Rightarrow}$ if there is a sequence of autonomous transitions $f \stackrel{\mu_1}{\Longrightarrow} f_1 \stackrel{\mu_2}{\Longrightarrow} \cdots \stackrel{\mu_n}{\Longrightarrow} f_n$ ($s = \mu_1, \ldots, \mu_n$), and likewise for infinite traces.

Lemma 1.2.22. *Given operations $g, \widetilde{f}$ and a (finite) trace s:*

1. *$g \circ \widetilde{f}^\infty \stackrel{s}{\Rightarrow}$ implies there is n such that $g \circ \widetilde{f}^n \stackrel{s}{\Rightarrow}$.*
2. *$g \circ \widetilde{f}^n \stackrel{s}{\Rightarrow}_{\widetilde{f}}$ implies $g \circ \widetilde{f}^\infty \stackrel{s}{\Rightarrow}$.*

Proof. 1. Immediate from Lemma 1.2.13.

2. We proceed by induction over the length of s, using Lemma 1.2.12.

□

All theorems obtained for $\approx$ can be adapted to $\approx_{tr}$, with similar proofs. As an example, Theorem 1.2.16 becomes (it is also possible to adapt Theorem 1.2.21, with the same hypotheses – see Appendix C.3.2 for such a proof, in a variant of this abstract setting):

Theorem 1.2.23. *Let $\widetilde{f} \in \mathbf{C}$ be operations of arity n such that operations in $\mathbf{C}$ respect $\approx_{tr}$. If the fixpoints $\widetilde{f}^\infty$ do not diverge, then the equation $\widetilde{X} = \widetilde{f}(\widetilde{X})$ has at most one solution for $\approx_{tr}$.*

Proof. For simplicity, we consider the proof only in the case of a single equation $X = f(X)$ (extension to systems of equations is as in proof of Theorem 1.2.16).

We proceed by showing that $x \subseteq_{tr} f(x)$ implies $x \subseteq_{tr} f^\infty$, and then that $f(x) \subseteq_{tr} x$ implies $f^\infty \subseteq_{tr} x$. This indeed gives that $x \approx_{tr} f(x)$ implies $x \approx_{tr} f^\infty$; hence the equation has at most one solution (f^∞ does not belong to the LTS, hence it is not a solution).

1. $f(x) \subseteq_{tr} x$ implies $f^\infty \subseteq_{tr} x$. For this part, the absence of divergence hypothesis is not needed.

   Let s be a trace, and assume $f^\infty \stackrel{s}{\Rightarrow}$. By Lemma 1.2.22, there is n such that $f^n \stackrel{s}{\Rightarrow}$. Hence, $f^n(x) \stackrel{s}{\Rightarrow}$. Since $f(x) \subseteq_{tr} x$ and f respects $\subseteq_{tr}$, we have that $f^n(x) \subseteq_{tr} f^{n-1}(x) \subseteq_{tr} \ldots \subseteq_{tr} x$. Hence, $x \stackrel{s}{\Rightarrow}$, and $f^\infty \subseteq_{tr} x$.

2. $x \subseteq_{tr} f(x)$ implies $x \subseteq_{tr} f^\infty$. This part of the proof is very similar to the proofs of Theorems 1.1.9 and 1.2.21.

   Assume $x \subseteq_{tr} f(x)$, and $x \stackrel{s}{\Rightarrow}$. We want to show that $\mathbf{id} \stackrel{s}{\Rightarrow}_{\widetilde{f}}$, then apply Lemma 1.2.22; this would show $f^\infty \stackrel{s}{\Rightarrow}$. To that end, we build a strictly increasing sequence of (autonomous) transitions $\mathbf{id} \stackrel{s_n}{\Rightarrow}_{\widetilde{f}} g_n$, such that $g_n(x) \stackrel{s'_n}{\Longrightarrow}$ and such that $s = s_n s'_n$ for all n (s_n and s'_n are traces whose concatenation is s). Strictly increasing means that the transition $\mathbf{id} \stackrel{s_n}{\Rightarrow}_{\widetilde{f}} g_n$ is a strict prefix of the transition $\mathbf{id} \stackrel{s_{n+1}}{\Longrightarrow}_{\widetilde{f}} g_{n+1}$ This construction will have to stop, otherwise we build a divergence.

   Construction of the sequence. We initialise with the empty trace from $\mathbf{id}$. Indeed, we have $\mathbf{id}(x) \Rightarrow \mathbf{id}(x) \stackrel{s}{\Rightarrow}$, and s is indeed the concatenation of itself with the empty trace.

   Then, at step n, suppose we have, for instance, $\mathbf{id} \stackrel{s_n}{\Rightarrow}_{\widetilde{f}} g_n$, and $g_n(x) \stackrel{s'_n}{\Longrightarrow}$.

- If s'_n is the empty trace, we stop. We have in this case $\mathbf{id} \overset{s}{\Rightarrow}_{\widetilde{f}}$, which concludes.
- Otherwise, choose k minimal such that $g_n \circ f^k$ is autonomous (exists by Lemma 1.2.10). The operations respect $\subseteq_{tr}$, and by hypothesis $x \subseteq_{tr} f(x)$, therefore $g_n(x) \subseteq_{tr} g_n \circ f^k(x)$. From there, $g_n \circ f^k(x) \overset{s'_n}{\Longrightarrow}$.

  s'_n is not the empty sequence, thus $\overset{s'_n}{\Longrightarrow}$ is not empty either. $g_n \circ f^k$ is autonomous; we take g_{n+1} such that $g_n \circ f^k \xrightarrow{\mu} g_{n+1}$ is the first transition of $\overset{s'_n}{\Longrightarrow}$. If $\mu \neq \tau$, $s_{n+1} = s_n\mu$ and s'_{n+1} is s'_n with the first action removed, otherwise $s_{n+1} = s_n$ and $s'_{n+1} = s'_n$.

  Then, we indeed have $\mathbf{id} \overset{s_{n+1}}{\Longrightarrow}_{\widetilde{f}} g_{n+1}$ and $g_{n+1}(x) \overset{s'_{n+1}}{\Longrightarrow}$ (by construction) wehere $s = s_{n+1}s'_{n+1}$.

Suppose now that the construction given above never stops. To make the argument clearer, we reason up to $=_f$ (we identify all operations that are $=_f$). We know that $g_n \xrightarrow{\mu_n}_{\widetilde{f}} g_{n+1}$ for some μ_n, therefore, by applying Lemma 1.2.12 as many times as needed (just as in Lemma 1.2.22), we get that $g_n \circ f^\infty \overset{\mu_n}{\Longrightarrow} g_{n+1} \circ f^\infty$.

This gives an infinite sequence of transitions starting from f^∞: $f^\infty \overset{\mu_0}{\Longrightarrow} g_1 \circ f^\infty \overset{\mu_1}{\Longrightarrow} \dots$. We observe that in the latter sequence, every step involves at least one transition. Moreover, the sequence $s''_1 s''_2 \dots$, when removing all τs, is a prefix of s. Therefore there is a finite number of visible actions occurring in this infinite sequence. Therefore f^∞ is divergent.

□

In contrast, the theorems fail for *infinitary trace equivalence*, $\approx^\infty_{tr}$ (whereby two processes are equated if they have the same traces, including the infinite ones), for the same reason why the 'unique solution of contraction' technique fails in this case [San15]. As a counterexample, we consider equation $X = a + a.\mathbf{X}$, whose syntactic solution has no divergences. The process $P \overset{\triangle}{=} \sum_{n>0} a^n$ is a solution, yet it is not $\approx^\infty_{tr}$-equivalent to the syntactic solution of the equation, because the syntactic solution has an infinite trace involving a transitions. This phenomenon is due to the presence of infinitary observables.

1.2.3 Preorders

We show how the theory for equivalences can be transported onto *preorders*. This means moving to *systems of pre-equations*, $\{X_i \leq E_i\}_{i \in I}$. With preorders, our theorems have a different shape: we do not use pre-equations to reason about unique solution – we expect interesting pre-equations to have many solutions, some of which may be incomparable with each other. We rather derive theorems to prove that, in a given preorder, any solution of a pre-equation is below its syntactic solution.

In the LTS we consider, an equation does not always have a solution. We thus extend the original LTS with the transitions corresponding to the autonomous transitions of the syntactic solution f^∞.

We write $\subseteq_{tr}$ for trace inclusion, $\subseteq_{tr^\infty}$ for infinitary trace inclusion, and $\leq_s$ for weak simulation. These preorders are standard from the literature [vG90].

In the abstract setting, the body of the pre-equations are functions. Then the theorems give us conditions under which, given a pre-equation $X \leq f(X)$ and a behavioural preorder $\preceq$, a solution r, i.e., a state for which $r \preceq f(r)$ holds, is below the syntactic solution f^∞. We present the counterpart of Theorem 1.2.21; other theorems are transported in a similar manner, both the statements and the proofs.

Given a preorder $\leq$, we say that $\widetilde{x}$ is a *pre-fixed point for* $\leq$ of operations $\widetilde{f}$ if $\widetilde{f}(\widetilde{x}) \leq \widetilde{x}$; similarly, $\widetilde{x}$ is a *post-fixed point for* $\leq$ if $\widetilde{x} \leq \widetilde{f}(\widetilde{x})$.

Theorem 1.2.24. *Choose a preorder $\leq \in \{\subseteq_{tr}, \subseteq_{tr^\infty}, \leq_s\}$. Let $\widetilde{f} \in \mathbf{C}$ be operations of arity n, such that $\mathbf{C}$ respect $\leq$. If the fixpoints $\widetilde{f}^\infty$ do not diverge, whenever $\widetilde{x} \leq \widetilde{f}(\widetilde{x})$ we also have $\widetilde{x} \leq \widetilde{f}^\infty$, for any states $\widetilde{x}$.*

Proof. 1. For $\subseteq_{tr}$, the proof is given as the item 2 of the proof of Theorem 1.2.23.

2. For $\subseteq_{tr^\infty}$, the proof is very similar to the previous proof: simply consider infinite traces, and rather that disjunct over whether or not the construction stops (the construction cannot stop), disjunct over whether the full trace is captured, or only τ actions occur from a moment in the construction.

3. For $\leq_s$ The proof is strictly included in the proofs of Theorems 1.1.9 or 1.2.21.

□

Theorem 1.2.24 intuitively says that the syntactic solution of a pre-equation is *maximal* among all solutions. Note that, in contrast with equations, Theorem 1.2.24 and the theory of pre-equations also work for *infinitary trace inclusion.*

The opposite direction for pre-equations, namely $\{X_i \geq E_i\}_{i \in I}$ is less interesting; it means that the syntactic solution is *minimal* among the solutions. This property is usually easy to obtain for a behavioural preorder, and we do not need hypotheses such as autonomy, well-behavedness, or non divergence, as stated in the following proposition:

Proposition 1.2.25. *Let $\widetilde{f} \in C$ be operations in a clone $\mathbf{C}$, $\leq \in \{\subseteq_{tr}, \leq_s\}$, and $\widetilde{x}$ such that $\widetilde{f}(\widetilde{x}) \leq \widetilde{x}$ and $\mathbf{C}$ respects $\leq$. Then, $\widetilde{f}^\infty \leq \widetilde{x}$.*

Proof. 1. For $\subseteq_{tr}$, the proof is given as item 1 of the proof of Theorem 1.2.23.

2. For $\leq_s$, the proof is very similar to the above proof for $\subseteq_{tr}$.

□

However, Proposition 1.2.25 may fail for preorders with infinitary observables, such as infinitary trace inclusion. This is why the theory of equations fails for infinitary trace equivalence. Indeed, if P is a solution for the equation $E \subseteq_{tr^\infty} \mathbf{X}$ (whether or not E is divergence-free), then $K_E \subseteq_{tr^\infty} P$ does not necessarily hold. The equation $X = a + a.X$, which is discussed after Theorem 1.2.23, provides a counter-example to illustrate this observation.

Remark 1.2.26. Suppose that x and y are such that $x \leq f(x)$ and $y \leq x$. Then we do not necessarily have $y \leq f(y)$. Take for instance, in CCS, $P = a.\,b$, $Q = a \mid b$ and $E = a.\,X + b.\,X$. Then $Q \leq_s E[Q]$, and $P \leq_s Q$, however $P \not\leq_s E[P]$.

By combining Theorem 1.2.24 and Proposition 1.2.25, we obtain a symmetrical version of the unique-solution theorem. Its statement is arguably more useful to derive a preorder relation, essentially because the only requirement involving the syntactic solution of $\mathcal{E}$ is about divergences.

Theorem 1.2.27. *Choose a preorder $\leq \in \{\subseteq_{tr}, \subseteq_{tr^\infty}, \leq_s\}$. Let $\widetilde{f} \in \mathbf{C}$ be operations of arity n, such that $\mathbf{C}$ respect $\leq$. If the fixpoints $\widetilde{f}^\infty$ do not diverge, whenever $\widetilde{x} \leq \widetilde{f}(\widetilde{x})$ and $\widetilde{f}(\widetilde{y}) \leq \widetilde{y}$ we also have $\widetilde{x} \leq \widetilde{y}$, for any states $\widetilde{x}, \widetilde{y}$.*

Proof. We apply Theorem 1.2.24 to $\widetilde{x}$ and Proposition 1.2.25 to $\widetilde{y}$, which gives $\widetilde{x} \leq \widetilde{f}^\infty \leq \widetilde{y}$. □

1.3 Rule formats

A way to instantiate the results in Sections 1.3.1 and 1.2.2 is to consider *rule formats* [AFV01, MRG07]. These provide a specification of the expected shape of the SOS rules used to describe the constructs of a language. To fit a rule format into the abstract formulation of the theory from Section 1.3.1, we view the constructs of a language as functions on the states of the LTS (the terms or processes of the language).

An quick overview of term algebras, TSS and rule formats are given in Section 0.4.3. For a complete survey of rule formats and for the relevant definitions, we refer the reader to [AFV01, MRG07, BIM88].

We first discuss some standard formats, such as GSOS [BIM88, vG05, FvG16], or the tyft/tyxt [GV92] formats. We also discuss their relationship with the notions of guardedness, and how look-aheads from the tyft/tyxt formats fit in our theory.

The relationship between the abstract setting from the previous section and syntactic methods to specify recursive behaviour (e.g., recursive constants in CCS) is studied by the means of the cool formats in Section 1.3 afterwards.

1.3.1 Autonomy and guardedness

In this section, we present an alternative formulation of this abstract framework, closer to the original formulation in [DHS17, DHS19]. Formal definitions that are only slightly different from [DHS17] are not given again here, but the whole development is accessible in Appendix C.3. While the derived technique is strictly weaker, it is enlightening about the role played by guardedness and the treatment of look-aheads.

In this formulation, unfolding $g \circ \widetilde{f}^\infty$ to $(g \circ \widetilde{f}) \circ \widetilde{f}^\infty$ does not produce a τ-transition, in the same way that in CCS, unfolding a constant K does not produce a τ.

Autonomous transitions for terms $g \circ \widetilde{f}^\infty$ are thus defined using the following rules:

$$\frac{g \xrightarrow{\mu} g'}{g \circ \widetilde{f}^{\infty} \xrightarrow{\mu} g' \circ \widetilde{f}^{\infty}}\ g \text{ autonomous} \qquad \frac{(g \circ \widetilde{f}) \circ \widetilde{f}^{\infty} \xrightarrow{\mu} F}{g \circ (\widetilde{f}^{\infty}) \xrightarrow{\mu} F}\ g \text{ not autonomous}$$

We disallow unnecessary unfoldings; these would duplicate transitions (the transitions of $g \circ \widetilde{f}$ duplicate those of g when g is autonomous). This method ensures that there is a perfect match between transitions of $\widetilde{f}^{\infty}$ and of the unfoldings f^n of f (otherwise, this would only hold for weak transitions).

For a transition to be performed, an autonomous operation must be reached through successive unfoldings of $\widetilde{f}$. Otherwise, the term is stuck. For instance, the CCS equation expression $E = X \mid \overline{a}$ would have no transition, whereas with the rules in the previous section, we would have $E^{\infty} \xrightarrow{\tau} \xrightarrow{\overline{a}} \dots$

Consider the equation $X = X$. It now is divergence-free. This means we have to consider operations that are *eventually autonomous*, as in CCS, whereas in the previous framework, such an hypothesis is not needed: operations that are not eventually autonomous always diverge. This is similar to the variant of CCS where constants perform a transition $K \xrightarrow{\tau} P$ (where $K \triangleq P$), capturing the guardedness hypothesis within the non-divergence one.

This guardedness hypothesis is not, however, sufficient. We must also demand that operations *respect autonomy*. Take, for instance, a single unary operation f. Eventually guarded means that there is n such that f^n is autonomous. However, given an operation g, this does not mean that there is m such that $g \circ f^m$ is autonomous. Hence, we demand that if f is autonomous, then so is $g \circ f$ (for any g in the clone). The symmetric hypothesis, i.e., that $g \circ f$ is autonomous whenever g is, always hold: indeed, transitions of $f(x)$ do not depend on x, hence transitions of $f(g(x))$ do not depend on either x or g.

We give a concrete example, illustrating the need for this hypothesis.

Example 1.3.1. Consider a language with three constructors: first, a prefix constructor, like the prefix of CCS, and with the same rule. We assume that we can use the prefix constructor with at least two distinct labels, a and b. Second, a constant $\mathbf{0}$, that has no rule (as in CCS). Third, a constructor `forward` with the following rule:

$$\frac{x \xrightarrow{\mu'} y \quad y \xrightarrow{\mu} z}{\texttt{forward}(x) \xrightarrow{\mu} z}$$

Consider the equation

$$X = a.\,\texttt{forward}(X) \tag{1.4}$$

It is autonomous or (strongly) guarded, and has no divergence: write $f : x \mapsto a.\,\texttt{forward}(x)$, then $f^{\infty} \xrightarrow{a} \texttt{forward}(f^{\infty})$, which has no transitions (intuitively, $\texttt{forward}(a.\,\texttt{forward}(a.\dots))$ has no transitions, because the `forward` construct always skip the visible input, and a possible transition is never reached). However, it does not enjoy unique solution for $\approx$: both $a.\,\mathbf{0}$ and $a.\,b.\,\mathbf{0}$ are solution.

A variant of this example is discussed at the end of this section (Example 1.3.2).

In Example 1.3.1, the construct `forward` breaks autonomy: assuming f is autonomous, $\texttt{forward} \circ f$ might not be autonomous.

The rule defining this construct uses look-aheads. A look-ahead allows one to write rules that 'look into the future' (a transition is allowed if certain sequences of actions are possible); this breaks autonomy. Look-aheads are sometimes forbidden by rule formats, such as GSOS; however look-aheads are possible in the tyft/tyxt formats [GV92], for instance. And indeed, the small language defined in Example 1.3.1 fits the tyft format.

This means the theory presented here may not be applied to these formats, nor to any other format that allows look-aheads; on the other hand, Theorem 1.2.21 can be applied to languages defined by such formats. This is explained by the fact that the non-divergence hypothesis in Theorem 1.2.21 is more discriminating, as there are more divergences, created by unfolding infinitely non-autonomous operations. Theorem 1.2.21 thus captures the fact that equation 1.4 does not have a unique solution: its syntactic solution diverges.

On the other hand, the equation

$$X = a.\, a.\, \texttt{forward}(X) \tag{1.5}$$

does have a unique solution; and while this can be shown using Theorem 1.2.21, applying the alternative theory is not possible: operations from the clone must respect autonomy (or, at the very least, the reducts of the operations defining the equations must respect autonomy). However, here, the operation $x \mapsto \texttt{forward}(x)$ does not.

The idea behind the rule

$$\frac{}{g \circ (\widetilde{f}^{\infty}) \xrightarrow{\tau} (g \circ \widetilde{f}) \circ \widetilde{f}^{\infty}}\ g \text{ not autonomous} \tag{1.2}$$

is to add τ-transitions, as many as needed to capture all equations that do not have a unique solution. In CCS with the τ-unfolding rule, this rule is admissible. Indeed, any construct `op` that is not autonomous (for instance, parallel composition or restriction) verifies that if $P \xrightarrow{\tau} P'$, then $\texttt{op}(P) \xrightarrow{\tau} \texttt{op}(P')$. Such a construct can then be combined with the unfolding rule

$$\frac{}{K \xrightarrow{\tau} P} K \triangleq P$$

If a language were to be enriched with similar rule, any non-autonomous construct would need to verify a similar property. In other words, the rule

$$\frac{x \xrightarrow{\tau} y}{\texttt{op}(x) \xrightarrow{\tau} \texttt{op}(y)}$$

Would have to be admissible for any non-autonomous construct `op` (or similarly for any context) . This is the spirit of the formats presented in Section 1.3.2, similar also to the cool formats for congruence for weak bisimilarity [vG05].

A possible objection to Example 1.3.1 is precisely that it does not follow this discipline. Indeed, the `forward` construct allows to 'count' the number of τ-transitions of a state; this is not in the spirit of weak equivalences. We thus present a variant of Example 1.3.1, more along the lines of formats for weak equivalences. It still fits within the tyft format.

Example 1.3.2. We consider a language similar to that of Example 1.3.1; however, there is now a construct $\texttt{forward}_a$ for each (visible) label a. We still assume there are at least two distinct visible labels a and b. We now consider the following rules for $\texttt{forward}_a$ (for each label a):

$$\frac{x \xrightarrow{a} y \quad y \xrightarrow{\mu} z}{\texttt{forward}_a(x) \xrightarrow{\mu} z} \qquad \frac{x \xrightarrow{\tau} y}{\texttt{forward}_a(x) \xrightarrow{\tau} \texttt{forward}_a(y)}$$

As the `forward` constructs are not autonomous, we ensure that they allow τ-transitions to occur.

Again, the equation $X = a.\texttt{forward}_a(X)$ is autonomous and has no divergences, but does not enjoy unique solution for $\approx$, for similar reasons.

Remark 1.3.3. The tyft/tyxt formats, and, more generally, languages with look-aheads, are an example of languages where up to context is sometimes sound, but never compatible, as explained in Section 0.4.2.3.

1.3.1.1 GSOS format

We now discuss one of the most common formats, GSOS [BIM88, vG05, FvG16]. This format guarantees that the constructs of the language preserve autonomy. This allows to show that the clone of contexts in a GSOS language constitute a set of operations that respect autonomy (when seen as functions from terms to terms).

We now use x, y for *state variables*, and t, u for terms that are part of the considered GSOS language (i.e., for elements of the set of states). We use c, d for contexts, seen as functions from the set of states to itself. We write `op` for a construct of the language (a function symbol).

We recall the format of GSOS rules below, slightly modified in order to use contexts in place of terms:

$$\frac{\{x_i \xrightarrow{\mu_{i,j}} y_{i,j} \mid i \in I,\ 1 \leq j \leq m_i\} \cup \{x_j \xrightarrow{\mu'_{j,k}} \not\rightarrow \mid j \in J,\ 1 \leq k \leq n_j\}}{\texttt{op}(\widetilde{x}) \xrightarrow{\mu} c(\widetilde{x}, \widetilde{y})}$$

I, J are fixed subsets of $[1, n]$, where n is the length of $\widetilde{x}$. For $i \in I$ (resp. $j \in J$), m_i (resp. n_j) is a fixed integer. $\widetilde{y}$ is the list consisting of all $y_{i,j}$ for $i \in I$ and $1 \leq j \leq m_j$, as well as all $y_{j,k}$ for $j \in J$ and $1 \leq k \leq n_j$. c is a context belonging to the language.

Lemma 1.3.4. *The clone of contexts of a language defined within the GSOS format is such that composition preserves autonomy (on this set).*

Proof. Let $\mathcal{L}$ be such a language. We reason by induction over the contexts $\mathbf{C}$ (which are defined inductively using the constructs of $\mathcal{L}$).

- The empty context (i.e., the 'hole', corresponding to the identity function $\mathbf{id}$) preserves autonomy: if $c \in \mathbf{C}$ is autonomous, then so is $\mathbf{id} \circ c = c$.

- Consider a n-ary construct of the language (a function symbol), $\mathtt{op}$, and a context $c = \mathtt{op}(c_1, \ldots, c_n)$, where the c_i's are unary contexts.

  We have to show that so is $c \circ c'$, for c' autonomous. Consider for this a transition $c \circ c'(t) \xrightarrow{\mu} u$ $(t, u \in \mathcal{L})$, we prove that this transition is autonomous. The last rule of the derivation tree of this transition is an instance of a GSOS rule as seen above:

$$\frac{\{x_i \xrightarrow{\mu_{i,j}} y_{i,j} \mid i \in I,\ 1 \leq j \leq m_i\} \cup \{x_j \xrightarrow{\mu'_{j,k}} \not\mid \ j \in J,\ 1 \leq k \leq n_j\}}{\mathtt{op}(\widetilde{x}) \xrightarrow{\mu} c_0(\widetilde{x}, \widetilde{y})}$$

  (I, J and the m_is and n_js being fixed, as well as c_0).

  By definition $c \circ c'(t) = \mathtt{op}(c_1 \circ c'(t), \ldots, c_n \circ c'(t))$. Therefore the x_i must be instantiated by the terms $c_i \circ c'(t)$. Hence there are $u_{i,j} \in \mathcal{L}$ such that $c_i \circ c'(t) \xrightarrow{\mu_{i,j}} u_{i,j}$ for $i \in I,\ 1 \leq j \leq m_i$. As well, $c_j \circ c'(t) \xrightarrow{\mu_{j,k}} \not\mid$ for $j \in J,\ 1 \leq k \leq n_j$. Lastly, writing $u = c_0(\widetilde{c} \circ c'(t), \widetilde{u})$ and $c \circ c'(t) \xrightarrow{\mu} c_0(\widetilde{c} \circ c'(t), \widetilde{u})$.

  By induction hypothesis, each of the c_i's preserves autonomy under composition on $\mathcal{O}$, hence $c_i \circ c'$ is autonomous for all i. Hence there are autonomous transitions $c_i \circ c' \xrightarrow{\mu_{i,j}} d_{i,j}$, where $d_{i,j}$ are contexts such that $u_{i,j} = d_{i,j}(t)$.

  Furthermore, for any $t' \in \mathcal{L}$, $c_j \circ c'(t') \xrightarrow{\mu_{j,k}} \not\mid$: if there was such a t', by autonomy of $c_j \circ c'$, we would have $c_j \circ c'(t) \xrightarrow{\mu_{j,k}}$.

  Therefore, for any $t' \in \mathcal{L}$, the premises of the above GSOS rule hold; hence we can deduce $c \circ c'(t') \xrightarrow{\mu} c_0(\widetilde{c} \circ c'(t'), \widetilde{d}(t))$. Thus there is an autonomous transition $c \circ c' \xrightarrow{\mu} c_0(\widetilde{c} \circ c', \widetilde{d})$, and $u = (c_0(\widetilde{c} \circ c', \widetilde{d}))(t)$. The transition $c \circ c'(t) \xrightarrow{\mu} u$ is indeed autonomous. This concludes the proof.

  □

1.3.2 Loosely cool GSOS formats

In this section we present a new format, a generalization of the GSOS cool formats discussed in Section 0.4.3.2, tailored specifically for our unique solution technique. It illustrates how to relate the arbitrary rules for unfolding in Section 1.2 with concrete, syntactic methods for defining recursive processes. Non-diverging equations written in a language defined within this format enjoy the unique-solution property.

Cool formats originate from [Blo95, UP02], but are here presented in a fashion similar to [vG05, vG11].

We say that variable x_i is at position i in the term $\mathtt{op}(x_1, \ldots, x_n)$ (if $i \leq n$). Patience rules are defined similarly to [vG05, vG11]:

Definition 1.3.5 (Patience rule)**.** A rule of the shape

$$\frac{x_i \xrightarrow{\tau} y}{\mathtt{op}(\widetilde{x}) \xrightarrow{\tau} \mathtt{op}(\widetilde{x})\{y/x_i\}}$$

is called a *patience rule* for $\mathtt{op}$ at position i.

To define the WB cool GSOS formats (and other variants), Van Glabeek uses the notion of *active* and *receiving* variables. A variable is active if it is both in the source of a rule and the source of a premise of the same rule, while it is receiving when it is both the target of a rule and on the right-hand side of a premise of that same rule. We want to allow for more general formats, and we do not intend that our formats enforce the congruence property, we thus are only interested in *active* variables – more precisely, in variables that appear both in the source of a rule and in the left-hand side of a premise. We say that an operator with such a variable depends on this variable. Indeed, receiving variables have no impact on guardedness (i.e., to know whether a transition is possible, we only need to look at the source of a rule, not at its target).

Definition 1.3.6 (Dependence)**.** Consider a TSS with signature Σ, and some operator $\mathtt{op} \in \Sigma$ of arity n. We say that $\mathtt{op}$ depends on position $i \leq n$ if there is a rule with source $\mathtt{op}(\widetilde{x})$, such that x_i appears in some premise of this rule.

This notion of dependence captures a local version of autonomy: if $\mathtt{op}$ does not depend on any variable, then it is autonomous. A more general statement is shown in Lemma 1.3.12.

As in [vG05], we impose that any operator that depends on a variable or position (in [vG05], that has an active or receiving variable) has a patience rule in that same position. This allows for unfolding transitions $K \xrightarrow{\tau} P$ (when $K \stackrel{\triangle}{=} P$) to percolate through non-guarded operators.

Definition 1.3.7 (Loosely cool format)**.** A language is *loosely cool* if whenever an operator $\mathtt{op}$ depends on position k, then is a patience rule for $\mathtt{op}$ at that same position k.

The WB cool GSOS format is a specialization of the loosely cool GSOS format.

We now fix a language $\mathcal{L}$, defined within a loosely cool GSOS format. To extend it with recursive terms, we apply a method similar to the one used in CCS. We however use the alternate rule for unfoldings (with explicit τ-transitions), as in previous sections.

Definition 1.3.8 (Constants)**.** We extend $\mathcal{L}$ with an infinite set of *constant names*, $K, K' \ldots$ Each constant is given a definition $K \stackrel{\text{def}}{=} P$, where P is a process (constant names might therefore appear in P). For any constant name K, the following rule is added to the TSS of the language:

$$\frac{}{K \xrightarrow{\tau} P} K \stackrel{\triangle}{=} P$$

Remark 1.3.9 (Infinite unfoldings vs. fixpoints.)**.** We will use the notion of fixpoints from Section 1.2 (that relies on the semantic notion of autonomy), written E_i^∞ for a system of equations $\widetilde{E}$; this is different from the notions of syntactic solutions, defined as the recursive constants $K_{\widetilde{E},i}$, from Section 1.1.1. The point of this section is precisely to demonstrate that these two object correspond (for weak bisimilarity, and have the same divergences), assuming adequate hypothesis on the language.

When considering a context, seen as a state operation, with multiple arguments (multiple types of holes), we have to discriminate between arguments at which unfoldings can occur, and arguments at which they cannot. We thus introduce a notion of *partial autonomy*, representing the autonomy relative to a specific argument or variable of a state operation. To that end, we first define partial composition of functions.

Definition 1.3.10 (Partial composition)**.** Let f and g be two operations of arity n. Partial composition of f and g along i, written $g \circ_i f$, is defined as the operation of arity n, $g \circ \widetilde{f}$, where $f_i \stackrel{\triangle}{=} f$ and $f_j \stackrel{\triangle}{=} \pi_j^n$ for all $j \neq i$.

Constant operations. Given a set S, a constant operation $S \to S$ is an operation of arity 1 that maps any value to some fixed value x, $y \mapsto x$. We write x for the constant operation $y \mapsto x$ when it is clear from the context that x is a constant operation. Thus, we write $f \circ (x_1, \dots, x_{i-1}, \mathbf{id}, x_{i+1}, \dots)$ or $f(x_1, \dots, x_{i-1}, \mathbf{id}, x_{i+1}, \dots)$ for the following operation $S \to S$:

$$x \mapsto f(x_1, \dots, x_{i-1}, x, x_{i+1}, \dots)$$

Definition 1.3.11 (Partial autonomy)**.** Let f be a state operation in a clone on S containing all constant operations $S \to S$. We say it is i-autonomous if, forall $\widetilde{x} \in S^{n-1}$, $f(x_1, \dots, x_{i-1}, \mathbf{id}, x_{i+1}, \dots) : S \to S$ is autonomous.

We can now state the property of operators that do not depend on a specific variable or position: these are autonomous relative to that position.

Lemma 1.3.12. *If a constructor* $\mathtt{op}$ *of arity* n *does not depend on position* i*, then the operation* $\widetilde{x} \mapsto \mathtt{op}(\widetilde{x})$ *is* i*-autonomous.*

We now state general properties of partial composition.

Lemma 1.3.13. *Let* $\mathbf{C}$ *be a clone on* S *containing all constants operations of* S*. Let* g*,* $\widetilde{f}$*, be state operations of arity* n *in* $\mathbf{C}$*. The following hold*

1. *If* $i \neq j$*, then* $g \circ_i f_i \circ_j f_j = g \circ_j f_j \circ_i f_i$
2. $g \circ_1 f_1 \circ_2 f_2 \cdots = g \circ \widetilde{f}$
3. *If* g *is autonomous, then it is* i*-autonomous for all* i
4. *If* $g \circ_i f_i$ *is autonomous, then* $g \circ \widetilde{f}$ *is autonomous; likewise, if* $g \circ_{i_1} f_{i_1} \circ_{i_2} f_{i_2} \dots$ *is autonomous for some pairwise distinct* $i_1, i_2, \dots$*, then* $g \circ \widetilde{f}$ *is autonomous.*

Proof. 1 and 2 are straightforward.
3. Assume g is autonomous. Fix i, and consider a transition $g(\widetilde{x}) \xrightarrow{\mu} z$. Because g is autonomous, there is h such that $g \xrightarrow{\mu} h$ and $z = h(\widetilde{x})$. Now given some y, $g(\widetilde{x}\{y/x_i\}]) \xrightarrow{\mu} h(\widetilde{x}\{y/x_i\})$, hence the operation $g(\widetilde{x}\{\pi_i/x_i\}) : S \to S$ is autonomous.
4. First, remark that if g is autonomous, then $g \circ_i f$ is autonomous: if g is autonomous, then $g \circ \widetilde{f}$ is autonomous for all $\widetilde{f}$. Now assume $h = g \circ_{i_1} f_{i_1} \circ_{i_2} f_{i_2} \dots$ is autonomous. Write I for the set of indices $i_1, i_2, \dots$, and index as $j_1, j_2, \dots$ the set of indices not in I. Then $g \circ \widetilde{f} = h \circ_{j_1} f_{j_1} \circ_{j_2} f_{j_2} \dots$ is autonomous. □

We can now deduce that partial autonomy of composite functions can be derived from the partial autonomy of the components.

Lemma 1.3.14. *Let* $\mathtt{op}$ *be an operator in* $\mathcal{L}$*. If for all* j*, either* f_j *is* i*-autonomous, or* $\mathtt{op}$ *does not depend on* j*, then* $\mathtt{op} \circ \widetilde{f}$ *is* i*-autonomous.*

Proof. Fix some $\widetilde{x} \in S^{n-1}$. We show that $\mathtt{op} \circ \widetilde{f}(x_1, \dots x_{i-1}, \mathbf{id}, \dots, x_n) : S \to S$ is autonomous. Take $x_i, y \in S$, and consider a transition $\mathtt{op} \circ \widetilde{f}(\widetilde{x}) \xrightarrow{\mu} y$. This transition is an instance of a rule

$$\frac{H}{\mathtt{op}(\widetilde{z}) \xrightarrow{\mu} t}$$

with a substitution σ such that $\sigma(z_k) = f_k(\widetilde{x})$ and $t\sigma = y$. If $\mathtt{op}$ does not depend on j, z_j does not appear in H, hence $f_j(\widetilde{x})$ does not appear in $H\sigma$. Thus, if x_i appears in $H\sigma$, it is in a premise $f_j(\widetilde{x}) \xrightarrow{\mu} y'$, for a j such that f_j is i-autonomous; thus, there is h such that $f_j(x_1, \dots I \dots x_n) \xrightarrow{\mu} h$ and $h(x_i) = y'$. We can conclude from the fact that the rule is independent from x_i. □

We say that a context c is linear if each type of hole is used at most once in the syntax tree of c; likewise, we say that c is i-linear if the hole $[\cdot_i]$ is used at most once. An arbitrary context c can always be transformed into a linear context c' of different arity, such that for all $\widetilde{x}$, $c(\widetilde{x}) = c'(x_1, \dots x_1, x_2, \dots x_2, \dots x_n, \dots x_n)$.

We say that a rule is admissible if it can be derived from the rules of the language. We now show that to obtain a non-autonomous context, we have to combine non-autonomous operators, and thus, patience rules are admissible for non-autonomous contexts.

Lemma 1.3.15 (Autonomy and patience)**.** *Consider an* i*-linear context* c*. If* c *is not* i*-autonomous, then the rule*

$$\frac{x_i \xrightarrow{\tau} y}{c[\widetilde{x}] \xrightarrow{\tau} c[\widetilde{x}\{y/x_i\}]}$$

is admissible.

Proof. By induction on the syntax of C.

- If c is an empty context $[\cdot_j]$: c is not i-autonomous, thus $i = j$. Then the rule

$$\frac{x_i \xrightarrow{\tau} y}{x_i \xrightarrow{\tau} y}$$

  is trivially admissible.

- Assume c is a non-empty context, and is not i-autonomous. We write c as $\mathtt{op}(c_1, \ldots, c_n)$, where $\mathtt{op}$ is the constructor at the root of c. By the contrapositive of Lemma 1.3.14 there must be j such that $\mathtt{op}$ does not depend on j and c_j is not i-autonomous.

  By induction hypothesis, the rule

$$\frac{x_i \xrightarrow{\tau} y}{c_j[\widetilde{x}] \xrightarrow{\tau} c_j[\widetilde{x}\{y/x_i\}]}$$

  is admissible. By definition, there is a patience rule for position j in $\mathtt{op}$:

$$\frac{z_k \xrightarrow{\tau} y}{\mathtt{op}[\widetilde{z}] \xrightarrow{\tau} \mathtt{op}[\widetilde{z}\{y/z_k\}]}$$

  Because c is i-linear, only c_j may actually use x_i, thus, for $k \neq j$, $c_k[\widetilde{x}] = c_k[\widetilde{x}\{y/x_i\}]$. Thus we derive

$$\frac{\dfrac{x_i \xrightarrow{\tau} y}{c_j[\widetilde{x}] \xrightarrow{\tau} c_j[\widetilde{x}\{y/x_i\}]}}{\mathtt{op} \circ \widetilde{c}[\widetilde{x}] \xrightarrow{\tau} \mathtt{op} \circ \widetilde{c}[\widetilde{x}\{y/x_i\}]}$$

  This concludes. □

Lemma 1.3.15 can only be applied to linear contexts; however, as noted

We now derive the main result of this section, that allows, in conjunction with Theorem 1.2.16, to derive the unique solution theorem.

Lemma 1.3.16. *Let $\mathcal{L}$ be a language defined in a loosely cool GSOS format, extended with constants, and let $\widetilde{X} = \widetilde{E}[\widetilde{X}]$ be a system of equations of $\mathcal{L}$. If E_i^∞ diverges for some i, then its syntactic solutions diverge.*

Proof. Suppose $\widetilde{E}$ is such that E_i^∞ diverges:

$$E_i^\infty \xrightarrow{\mu_1} c_1 \circ E^\infty \xrightarrow{\mu_2} c_2 \circ E^\infty \ldots$$

and there is N such that for all $n \geq N$, $\mu_n = \tau$. We take c_0 to be the identity. We show that there is a sequence $(c'_n)_{n\in\mathbb{N}}$ such that $c_n =_{\widetilde{E}} c'_n$, where $=_{\widetilde{E}}$ is the smallest equivalence

relation relating c and c' whenever c' can be obtained by substituting some of the holes $[\cdot_i]$ in c by E_i. We also show that

$$\widetilde{K}_{\widetilde{E}} \xRightarrow{\mu_1} c'_1[\widetilde{K}_{\widetilde{E}}] \xRightarrow{\mu_2} c'_2[\widetilde{K}_{\widetilde{E}}] \dots$$

Furthermore, c_n is autonomous if and only if c'_n is autonomous, and likewise for partial autonomy relative to any i. We take c'_0 to be the identity as well. Now, set some $n \in \mathbb{N}$. We have

$$c_n \circ E^\infty \xrightarrow{\mu_{n+1}} c_{n+1} \circ E^\infty$$

- If c_n is autonomous, then $c_n \xrightarrow{\mu_{n+1}} c_{n+1}$ and so is c'_n; thus, $c'_n \xrightarrow{\mu_{n+1}} c'_{n+1}$ for $c'_{n+1} =_{\widetilde{E}} c_{n+1}$

- Otherwise, c_n is not autonomous, $\mu_{n+1} = \tau$, and $c_{n+1} = c_n \circ \widetilde{E}$. By Lemma 1.3.15, and linearising c_n, we deduce there is i such that there is a transition $c_n[\widetilde{K}_{\widetilde{E}}] \xRightarrow{\tau} d =_{\widetilde{E}} c_n \circ_i E_i[\widetilde{K}_{\widetilde{E}}]$. If d is autonomous, we take it to be c_{n+1}. Otherwise, there is j such that d is not j-autonomous; if i is the only such j, then we take $c'_{n+1} \stackrel{\triangle}{=} d$, and $c_{n+1} = c_n \circ \widetilde{E}$ is not autonomous; otherwise, $d[\widetilde{K}_{\widetilde{E}}] \xRightarrow{\tau} d'[\widetilde{K}_{\widetilde{E}}]$ for $d' =_{\widetilde{E}} c_n \circ_i E_i \circ_j E_j$. We perform this operation inductively, until there is no more such j, hence we take

  $$c'_{n+1} \stackrel{\triangle}{=} c_n \circ_{i_1} E_{i_1} \circ_{i_2} \dots$$

  Where the i_k are pairwise distinct. c'_{n+1} is autonomous (resp. j-autonomous) if and only if c_{n+1} is.

□

From Theorem 1.2.16 and Lemma 1.3.16, we derive the unique solution theorem.

Corollary 1.3.17 (Unique solution for loosely cool GSOS formats)**.** *Let $\mathcal{L}$ be a language defined in a loosely cool GSOS format, for which $\approx$ is a congruence, and let $\widetilde{X} = \widetilde{E}[\widetilde{X}]$ be a system of equations in $\mathcal{L}$. If its syntactic solutions do not diverge, then it has a unique solution.*

As the WB cool GSOS format enforces that bisimilarity is a congruence [vG05], we immediately derive from Corollary 1.3.17 the unique-solution property for the WB cool GSOS format.

Corollary 1.3.18 (Unique solution for WB cool GSOS)**.** *Let $\mathcal{L}$ be a language defined in the WB cool GSOS format, and let $\widetilde{X} = \widetilde{E}[\widetilde{X}]$ be a system of equations in $\mathcal{L}$. If its syntactic solutions do not diverge, then it has a unique solution.*

1.4 The unique solution technique in presence of name passing and higher order

1.4.1 The π-calculus and its subcalculi

1.4.1.1 Syntax of the π-calculus

In the π-calculus, the solutions of equations are parametric over their free names, i.e., they are functions from names to π-calculus processes. This allows us to work with closed agents, which makes the treatment of equations easier (in particular w.r.t. name capture). We call such parametric processes *abstractions*.

The instantiation of the parameters of an abstraction F is done via the *application* construct $F\langle\widetilde{a}\rangle$. We use P, Q for processes, F for abstractions. Processes and abstractions form the set of π-*agents* (or simply *agents*), ranged over by A. Small letters $a, b, \ldots, x, y, \ldots$ range over the infinite set of names. The grammar of the π-calculus is thus:

$$\begin{array}{lcll} A & := & P \mid F & \text{(agents)} \\ P & := & \mathbf{0} \mid a(\widetilde{b}).\,P \mid \overline{a}\langle\widetilde{b}\rangle.\,P \mid \boldsymbol{\nu} a\,P & \text{(processes)} \\ & & \quad \mid P_1 \mid P_2 \mid\, !a(\widetilde{b}).\,P \mid F\langle\widetilde{a}\rangle & \\ F & := & (\widetilde{a})\;P \mid K & \text{(abstractions)} \end{array}$$

$\mathbf{0}$, restriction and parallel composition are like in CCS. An input-prefixed process $a(\widetilde{b}).\,P$, where $\widetilde{b}$ has pairwise distinct components, waits for a tuple of names $\widetilde{c}$ to be sent along a and then behaves like $P\{\widetilde{c}/\widetilde{b}\}$, where $\{\widetilde{c}/\widetilde{b}\}$ is the simultaneous substitution of names $\widetilde{b}$ with names $\widetilde{c}$. An output particle $\overline{a}\langle\widetilde{b}\rangle$ emits names $\widetilde{b}$ at a. Replication could be avoided in the syntax since it can be encoded with recursion. For the sake equations using a replication above a variable, it is simpler to include it in the grammar (its semantics is simple).

We omit the operators of sum and matching, as they are not always considered fundamental operators of the π-calculus, and are not useful in the examples we study (particularly in Chapter 2). We assign parallel composition the lowest precedence among the operators. We refer to [Mil93] for detailed discussions on the operators of the language.

In prefixes $a(\widetilde{b})$ and $\overline{a}\langle\widetilde{b}\rangle$, we call a the *subject* and $\widetilde{b}$ the *object*. We use α to range over prefixes. When the tilde is empty, the surrounding parentheses or brackets in prefixes will be omitted. We often abbreviate $\alpha.\,\mathbf{0}$ as α, and $\boldsymbol{\nu} a\,\boldsymbol{\nu} b\,P$ as $(\boldsymbol{\nu} a, b)P$. An input prefix $a(\widetilde{b}).\,P$, a restriction $\boldsymbol{\nu} b\,P$, and an abstraction $(\widetilde{b})\;P$ are binders for names $\widetilde{b}$ and b, respectively, and give rise in the expected way to the definition of *free names* (fn) and *bound names* (bn) of a term or a prefix, and α-conversion. An agent is *name-closed* if it does not contain free names. (Since the number of recursive definitions may be infinite, some care is necessary in the definition of free names of an agent, using a least fixed-point construction.) As in the λ-calculus, we identify processes or actions which only differ in the choice of the bound names. The symbol $=$ stands for "syntactic identity modulo α-conversion". Sometimes, we

use $\stackrel{\text{def}}{=}$ as abbreviation mechanism, to assign a name to an expression to which we want to refer later.

We use constants, ranged over by K for writing recursive definitions. Each constant has a defining equation of the form $K \stackrel{\triangle}{=} (\widetilde{x})\ P$, where $(\widetilde{x})\ P$ is name-closed; $\widetilde{x}$ are the formal parameters of the constant (replaced by the actual parameters whenever the constant is used).

Since the calculus is polyadic, we assume a *sorting system* [Mil93] to avoid disagreements in the arities of the tuples of names carried by a given name and in applications of abstractions. We will not present the sorting system because it is not essential. The reader should take for granted that all agents described obey a sorting.

A *context* C of the π-calculus is a π-agent in which some subterms have been replaced by the hole $[\cdot]$ or, if the context is polyadic, with indexed holes $[\cdot]_1, \ldots, [\cdot]_n$. Then, $C[A]$ or $C[\widetilde{A}]$ is the agent resulting from replacing the holes with the agents A or $\widetilde{A}$. Holes in contexts have a sort too, as they could be in place of an abstraction.

Substitutions are of the form $\{\widetilde{b}/\widetilde{a}\}$, and are finite assignments of names to names. We use σ and ρ to range over substitutions. The application of a substitution σ to an expression H is written $H\sigma$. Substitutions have precedence over the operators of the language; $\sigma\rho$ is the composition of substitutions where σ is performed first, therefore $P\sigma\rho$ is $(P\sigma)\rho$.

The Barendregt convention allows us to assume that the application of a substitution does not affect bound names of expressions; similarly, when comparing the transitions of two processes, we assume that the bound names of the actions do not occur free in the processes. In a statement, we say that a name is *fresh* to mean that it is different from any other name which occurs in the statement or in objects of the statement like processes and substitutions.

Application redexes and normalised agents. A process of the form $((\widetilde{x})P)\langle\widetilde{a}\rangle$ is called an *application redex*; its contracted form is $P\{\widetilde{a}/\widetilde{x}\}$. An agent is *normalised* if it contains no application redex.

When reasoning on behaviours it is useful to assume that all expressions are normalised, in the above sense. Thus in the remainder of this thesis *we identify an agent with its normalised form.*

1.4.1.2 Operational semantics

The operational semantics of the π-calculus is standard [SW01]. Transitions of π-calculus processes are of the form $P \xrightarrow{\mu} P'$, where μ is an *action*. The grammar for actions is given by

$$\mu \quad := \quad a(\widetilde{b}) \ \Big|\ \boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle \ \Big|\ \tau \ .$$

- $P \xrightarrow{a(\widetilde{b})} P'$ is an input, where $\widetilde{b}$ are the names bound by the input prefix which is being fired,
- $P \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} P'$ is an output, where $\widetilde{d} \subseteq \widetilde{b}$ are private names extruded in the output, and

- $P \xrightarrow{\tau} P'$ is an internal action.

We abbreviate $\boldsymbol{\nu}\emptyset\,\overline{a}\langle\widetilde{b}\rangle$ as $\overline{a}\langle\widetilde{b}\rangle$. The occurrences of $\widetilde{b}$ in $a(\widetilde{b})$ and those of $\widetilde{d}$ in $\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle$ are bound; accordingly one defines the sets of bound names and free names of an action μ, respectively written $\mathrm{bn}(\mu)$ and $\mathsf{fn}(\mu)$. The set of all the names appearing in μ (both free and bound) is written $\mathrm{n}(\mu)$.

Figure 1.4 presents the transition rules for the π-calculus.

$$\frac{}{a(\widetilde{b}).\,P \xrightarrow{a(\widetilde{b})} P} \qquad \frac{}{!a(\widetilde{b}).\,P \xrightarrow{a(\widetilde{b})} !a(\widetilde{b}).\,P \mid P} \qquad \frac{}{\overline{a}\langle\widetilde{b}\rangle.\,P \xrightarrow{\overline{a}\langle\widetilde{b}\rangle} P}$$

$$\frac{P \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} P'}{\boldsymbol{\nu}n\,P \xrightarrow{\boldsymbol{\nu}(\{n\}\cup\widetilde{d})\,\overline{a}\langle\widetilde{b}\rangle} P'}\, n \in \widetilde{b} \qquad \frac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}n\,P \xrightarrow{\mu} \boldsymbol{\nu}n\,P'}\, d \notin \mathrm{n}(\mu) \qquad \frac{P \xrightarrow{a(\widetilde{b})} P' \qquad Q \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b'}\rangle} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu}\widetilde{d}\,(P'\{\widetilde{b'}/\widetilde{b}\} \mid Q')}$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\, \mathrm{bn}(\mu) \cap \mathsf{fn}(Q) = \emptyset \qquad \frac{F\langle\widetilde{a}\rangle \xrightarrow{\mu} P'}{K\langle\widetilde{a}\rangle \xrightarrow{\mu} P'} \text{ if } K \triangleq F$$

Figure 1.4: Labelled Transition Semantics for the π-calculus

$\Rightarrow$, $\stackrel{\mu}{\Rightarrow}$ and $\stackrel{\hat{\mu}}{\Rightarrow}$ are defined as in CCS. In bisimilarity or other behavioural relations for the π-calculus we consider, no name instantiation is used in the input clause or elsewhere; such relations are usually called *ground* [SW01]. Ground relations are not the standard behavioural equivalences for the π-calculus, as they are often not congruences. However, in the subcalculi of the π-calculus we consider (Aπ, Iπ, ALπ), ground bisimilarity is a congruence and coincides with barbed congruence (congruence breaks in the full π-calculus). The study of unique solutions of equations is also simpler when restricting to ground relations, in particular when checking divergences (see the proofs in Chapter 2). In the remainder, we omit the adjective 'ground'.

Example 1.4.1 (Ground divergence)**.** To illustrate the importance of the choice of LTS to the unique-solution technique, we give an example of a process that does not diverge for the ground LTS, but that diverges for another standard LTS of the π-calculus [SW01]. Consider the recursive constant:

$$K \triangleq (a,b)\ a(c).\,K\langle a,c\rangle \mid \overline{b}a$$

For $a \neq b$, the process $K\langle a,b\rangle$ does not diverge in the ground LTS: $K\langle a,b\rangle \xrightarrow{a(d)} K\langle a,d\rangle \mid \overline{b}a$ for d fresh, a synchronisation between the input on a and the output on b is never possible. However, if we consider the non-ground transition $K\langle a,b\rangle \xrightarrow{a(a)} K\langle a,a\rangle \mid \overline{b}a$, we then immediately have a divergence: $K\langle a,a\rangle \xrightarrow{\tau} K\langle a,a\rangle$.
The applications of the technique we consider in Chapter 2 have the same problem: for the

ground LTS, they have at worst innocuous divergences, however for other transition systems, they do have non-innocuous divergences.

Barbed congruence. As for CCS, the reference behavioural equivalence for the π-calculi usually is *barbed congruence*. We recall its definition, on a generic subset $\mathcal{L}$ of π-calculus processes A $\mathcal{L}$-*context* is a process of $\mathcal{L}$ with a single hole $[\cdot]$ in it.

We write $P \Downarrow_a$ if P can make an output action whose subject is a, possibly after some internal moves. We make only output observable because this is standard in asynchronous calculi. In the case of a synchronous calculus like Iπ, Definition 1.4.2 below yields synchronous barbed congruence, and adding also observability of inputs does not change the induced equivalence. More details on this are given in Section 2.2.8.

Definition 1.4.2 (Barbed congruence)**.** *Barbed bisimilarity* is the largest symmetric relation $\stackrel{\cdot}{\simeq}$ on π-calculus processes such that $P \stackrel{\cdot}{\simeq} Q$ implies:

1. If $P \Longrightarrow P'$ then there is Q' such that $Q \Longrightarrow Q'$ and $P' \stackrel{\cdot}{\simeq} Q'$.
2. For all a, $P \Downarrow_a$ iff $Q \Downarrow_a$.

Let $\mathcal{L}$ be a set of π-calculus agents, and $A, B \in \mathcal{L}$. We say that A and B are *barbed congruent in* $\mathcal{L}$, written $A \simeq^{\mathcal{L}} B$, if for each (well-sorted) $\mathcal{L}$-context C, it holds that $C[A] \stackrel{\cdot}{\simeq} C[B]$.

Remark 1.4.3. Barbed congruence has been uniformly defined on processes and abstractions (via a quantification over all process contexts). Usually, however, definitions will only be given for processes; it is then intended that they are extended to abstractions by requiring closure under ground parameters, i.e., by supplying fresh names as arguments.

As for all contextually-defined behavioural relations, so barbed congruence is hard to work with. To work with bisimilarity, we use a characterisation in terms of ground bisimilarity, with the (mild) condition that the processes are image-finite up to $\approx$. (We recall that the class of processes *image-finite up to* $\approx$ is the largest subset $\mathcal{IF}$ of π-calculus processes which is closed under transitions, and such that $P \in \mathcal{IF}$ implies that, for all actions μ, the set $\{P' \mid P \stackrel{\mu}{\Longrightarrow} P'\}$ quotiented by $\approx$ is finite. The definition is extended to abstractions as by Remark 1.4.3. All the agents in Chapter 2, including those obtained via an encoding of the λ-calculus, are image-finite up to $\approx$.

The definitions for bisimilarity and expansion are the same as for CCS (Definitions 0.2.2 and 0.3.11). We extend $\approx$ and $\succeq$ to abstractions, as per Remark 1.4.3: $F \approx G$ if $F\langle\widetilde{a}\rangle \approx G\langle\widetilde{a}\rangle$ for fresh $\widetilde{a}$, and, likewise, $F \succeq F'$ if $F\langle\widetilde{a}\rangle \succeq F'\langle\widetilde{a}\rangle$ for fresh $\widetilde{a}$.

1.4.1.3 Equations in the π-calculus

Equation expressions. We need variables to write equations. We use capital letters X, Y, Z for these variables and call them *equation variables*. The body of an equation is a name-closed abstraction possibly containing equation variables (that is, applications can also be of the form $X\langle\widetilde{a}\rangle$).

As for recursive definitions, so in equations the expression in the body is a closed abstraction. (Free names of equation expression and contexts are defined as for agents). Thus, the solutions of equations are abstractions. By imposing that equations must be closed abstraction, we can ignore some difficulties related to name capture.

We use E to range over expression bodies; and $\mathcal{E}$ to range over systems of equations, defined as follows. In all the definitions, the indexing set I can be infinite.

Solutions of a system of equations $\{X_i = E_i\}_{i\in I}$ is defined as for CCS (except that here equation expressions are closed abstractions, as well as solutions). Likewise, the syntactic solutions are the recursively defined constants $\mathrm{K}_{\widetilde{E},i} \stackrel{\triangle}{=} E_i[\widetilde{\mathrm{K}}_{\widetilde{E}}]$,(for each $i \in I$).

$E[\widetilde{F}]$ stands the abstraction obtained by replacing in E each occurence of the variable X_i with the abstraction F_i (assuming as usual that $\widetilde{F}$ and $\widetilde{X}$ have the same sort). This is syntactic replacement. However recall that we identify an agent with its normalised expression: hence replacing X with $(\widetilde{x})P$ in $X\langle\widetilde{a}\rangle$ amounts to replacing $X\langle\widetilde{a}\rangle$ with the process $P\{\widetilde{a}/\widetilde{x}\}$.

Example 1.4.4. If K is the syntactic solution of the equation $X = (a)\ !a(x).\, X\langle x\rangle$, then we have $K\langle b\rangle \xrightarrow{b(y)} K\langle y\rangle \mid K\langle b\rangle$.

Remark 1.4.5 (Contexts, equations and capture)**.** Equation expressions are essentially contexts. However, when considering equation expression transitions in the π-calculus, closed equation expressions applied to closed processes (using application $X\langle\widetilde{a}\rangle$ to explicit substitutions) behave differently from contexts: indeed, if $E\langle\widetilde{a}\rangle \xrightarrow{\mu} E'\langle\widetilde{a}\rangle$, then $E[F]\langle\widetilde{a}\rangle \xrightarrow{\mu} E'[F]\langle\widetilde{a}\rangle$ for all F; however, if $C \xrightarrow{\mu} C'$, we might need a substitution σ so that $C[P] \xrightarrow{\mu} C'[P\sigma]$; in other words, contexts must carry substitutions at their leafs, because of implicit name capture. This is what we attempt to explicit with parameters and abstractions.

1.4.1.4 Unique solution for Aπ

Theorems 1.1.7 and 1.1.9 for CCS can be adapted to the asynchronous π-calculus. The definitions concerning transitions and divergences are transported to Aπ as expected. In the case of an abstraction, one first has to instantiate the parameters with fresh names; thus F has a divergence if the process $F\langle\widetilde{a}\rangle$ has a divergence, where $\widetilde{a}$ are fresh names.

Theorem 1.4.6 (Unique solution in Aπ)**.** *A guarded system of equations whose syntactic solutions do not contain divergences has a unique solution for* $\approx$.

Proof. Because we only consider closed abstractions, all names are instantiated when performing a substitution of a variable by an abstraction. This allows us to use the same proof as for Theorem 1.1.9 in CCS. Some of the unique-solution proofs for subcalculi of the π-calculus are detailed in Appendix C. □

Theorem 1.4.7 (Unique solution with innocuous divergences in Aπ)**.** *A guarded system of equations whose syntactic solutions only have innocuous divergences has a unique solution for* $\approx$.

Proof. The proof of Theorem 1.4.6 has to be modified exactly as in the CCS case, where the proof of Theorem 1.1.7 is modified to establish Theorem 1.1.9. □

As in CCS, the guardedness condition can be removed if the rule for the unfolding of a constant produces a τ-transition.

We now state an analogue of Lemma 1.1.12 for Aπ, giving a sufficient condition to guarantee that a system of equations only has innocuous divergences. This condition is decidable, and sufficient for the applications in Chapter 2. We leave the study of finer conditions for future work.

Lemma 1.4.8. *Consider a well-sorted system of equations $\widetilde{X} = \widetilde{E}$ in Aπ (in particular, for each i, the sort of X_i and of E_i are the same).*

Suppose that there is a sort of names such that names of that sort are never used in subject output position. If for each i there is n_i such that in $(E_i)^{n_i}$, each equation variable occurs underneath an (input) prefix of that sort, then the system has only innocuous divergences.

As discussed in Section 0.4.4, the connection between techniques based on unique solution of equations and up-to-context techniques is less immediate in name-passing calculi. Indeed, up-to-context enhancements for the ground bisimilarity of the π-calculus require closure under name instantiation, even when ground bisimilarity is known to be preserved by substitutions (Open problem 0.4.13). Thus, when comparing two derivatives $C[P]$ and $C[Q]$, in general it is not sufficient that P and Q alone are in the candidate relation: one is required to include also all their closures under name substitutions (or, if the terms in the holes are abstractions, instantiation of their parameters with arbitrary tuples of names). In contrast, the two unique solution theorems above are 'purely ground': $F = (\widetilde{x})P$ is solution of an equation $X = (\widetilde{x})E$ if P and $E\{^F/_X\}$ are ground bisimilar – a single ground instance of the equation is evaluated.

Remark 1.4.9 (Unique solution in the full π-calculus)**.** In the full π-calculus, including the output prefix $\overline{a}\langle\widetilde{b}\rangle.\, P$, ground bisimilarity is not a congruence. One has therefore to use other forms of bisimilarity. The most used is *early bisimilarity*; correspondingly one uses an early transition system, where the parameters of inputs are instantiated with arbitrary (i.e., not necessarily fresh) names, as in $a(\widetilde{x}).\, P \xrightarrow{a\langle\widetilde{b}\rangle} P\{^{\widetilde{b}}/_{\widetilde{x}}\}$. Modulo the move to the early setting, the theory exposed for the asynchronous π-calculus also holds in the full π-calculus.

However, when considering a different LTS, such as the early LTS, divergences that did not exist in the ground LTS may arise, and as such, prevent Theorems 1.4.6 or 1.4.7 to be used. It is even possible that, by considering a different LTS, an equation that had a unique solution loses this property. We leave the study of this question for future work.

1.4.1.5 Iπ and ALπ

Other than Aπ, we will consider two subcalculi of the π-calculus: the Internal π-calculus (Iπ) [San96b, SW01], and the Asynchronous Local π-calculus (ALπ) [MS04, SW01]. These are obtained by imposing certain constraints on prefixes. The proofs of unique solution are

identical for Aπ, Iπ, and ALπ, and very similar to the case of CCS, for instance. Some of these proofs are detailed in Appendix C; to avoid redundancy, they are not detailed here.

Iπ. In Iπ, all outputs are bound. This is syntactically enforced by replacing the output construct with the bound-output construct $\overline{a}(\widetilde{b}).\, P$, which is an abbreviation for $\boldsymbol{\nu}\widetilde{b}\,\overline{a}\langle\widetilde{b}\rangle.\, P$. In all tuples (input, output, abstractions, applications) the components are pairwise distinct so to make sure that distinctions among names are preserved by reduction.

Theorem 1.4.10. *In Iπ, on agents that are image-finite up to $\approx$, barbed congruence and bisimilarity coincide.*

The encoding of the λ-calculus into Iπ yields processes that are image-finite up to $\approx$. Thus we can use bisimilarity as a proof technique for barbed congruence.

ALπ. The asynchronous local π-calculus, ALπ, is defined by enforcing that in an input $a(\widetilde{b}).\, P$, all names in $\widetilde{b}$ appear only in output position in P. Moreover, ALπ being *asynchronous*, output prefixes have no continuation; in the grammar of the π-calculus this corresponds to having only outputs of the form $\overline{a}\langle\widetilde{b}\rangle.\, \mathbf{0}$ (which we will simply write $\overline{a}\langle\widetilde{b}\rangle$). In AL$\pi$, to maintain the characterisation of barbed congruence as (ground) bisimilarity, the transition system has to be modified [MS04], in order to introduce additional processes (the 'links', sometimes also called forwarders) along the transitions. In Section 2.2.8, we present these modifications, and explain how they allow us to obtain for ALπ a property similar to that of Theorem 1.4.10 for Iπ.

We now present the unique-solution theorem for Iπ and ALπ. It is also possible to adapt Theorem 1.1.9, discriminating innocuous and non-innocuous divergences.

Theorem 1.4.11. *In Iπ and ALπ, a guarded system of equations whose syntactic solutions are divergence-free has unique solution for $\approx$.*

1.4.2 Unique solution in the Higher-order π-calculus

1.4.2.1 The Higher-Order π-calculus

In the syntax for HOπ we adopt, we use abstractions and concretions to represent input and output prefixes; we also use *first-order names*, i.e., names which carry nothing. These are opposed to *higher-order names*, i.e., names which are used to exchange processes. For the theory we shall develop, the presence of first-order names is not necessary, but makes the presentation of various results easier.

Thus, let $\mathcal{F}$ be the infinite set of first-order names, and $\mathcal{H}$ the infinite set of higher-order names. Then, $\overline{\mathcal{F}} \stackrel{\text{def}}{=} \{\overline{m} \mid m \in \mathcal{F}\}$, $\overline{\mathcal{H}} \stackrel{\text{def}}{=} \{\overline{a} \mid a \in \mathcal{H}\}$, $\mathcal{N} \stackrel{\text{def}}{=} \mathcal{F} \cup \mathcal{H}$ and $\overline{\mathcal{N}} \stackrel{\text{def}}{=} \overline{\mathcal{F}} \cup \overline{\mathcal{H}}$. The special symbol τ, which does not occur in $\mathcal{N}$ or $\overline{\mathcal{N}}$, denotes a silent step. We let μ range over $\mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$, and ℓ range over $\mathcal{F} \cup \overline{\mathcal{F}} \cup \{\tau\}$ (the set of CCS-like actions). We use symbols x, y, z for names in $\mathcal{N}$; symbols m, n for names in $\mathcal{F}$; and symbols a, b, c for names

in $\mathcal{H}$. We also assume an infinite set of *process variables*, ranged over by X, Y, Z, and a set *constant identifiers* (or simply *constants*) ranged over by K, H, to write recursively defined processes.

Definition 1.4.12. The syntactic categories of our language and their grammar are:

$$\begin{array}{rrcl} \text{Processes} & P & ::= & a.\,F \;\big|\; \overline{a}.\,C \;\big|\; \ell.\,P \;\big|\; P_1 \mid P_2 \;\big|\; \boldsymbol{\nu} a\,P \;\big|\; X \;\big|\; F \circ P \mid \mathbf{0} \\ \text{Abstractions} & F & ::= & (X)\,P \;\big|\; K \\ \text{Concretions} & C & ::= & \boldsymbol{\nu} x\,C \;\big|\; \langle P_1 \rangle P_2 \\ \text{Agents} & A & ::= & P \;\big|\; F \;\big|\; C \end{array}$$

A prefix $a.\,(X)\,P$ represents an input, in which the process received at a will replace X in P. Dually, $\overline{a}.\,\langle P_1 \rangle P_2$ is an output, in which P_1 is emitted at a and P_2 is the continuation. In an application $F \circ P$ process P is supposed to replace the formal parameter of the abstraction F. The remaining operators are the usual one of CCS and derived calculi. Symbols $P, Q, R, \ldots$ range over processes, F, G over abstractions, C, D over concretions, A, B over agents. Each constant K has a corresponding definition $K \stackrel{\triangle}{=} (X)\,P$ (as P may contain K, the behaviour may be recursive). Although in HOπ recursion can be derived (examples of this will be shown in Section 1.4.3), we take constants as primitives because they are useful when reasoning about equations. The calculus is monadic — (higher-order) names carry exactly one name; correspondingly, abstractions are parametrised over exactly one value. We stick to monadicity for simplicity of presentation.

We omit the parameter in constants when it is not needed, therefore simply writing $K \stackrel{\triangle}{=} P$ and simply using K as a process. We shall also sometimes use a special form of constants, namely the replication $!P$, that intuitively stands for an infinite number of copies of P in parallel; it can be written as the constant $K_P \stackrel{\triangle}{=} P \mid K_P$.

An application redex $((X)\,P) \circ Q$ can be normalised as $P\{{}^{Q}\!/_{X}\}$. An agent is *normalised* if all such application redexes have been contracted. Although the full application $F \circ P$ is often convenient, when it comes to reasoning on behaviours it is useful to assume that all expressions are normalised, in the above sense. Thus in the remainder of this section *we identify an agent with its normalised expression*. The application construct $F \circ P$ will play an important role in the treatment of equations in the following sections.

We shall only admit *standard concretions*, i.e., expressions $\boldsymbol{\nu} \widetilde{x} \langle P_1 \rangle P_2$ where names in $\widetilde{x}$ are pairwise distinct and $\widetilde{x} \subseteq \mathsf{fn}(P_1)$. Indeed, the remaining concretions have little significance: in $\boldsymbol{\nu} \widetilde{x}\, \langle Q \rangle P$, by alpha conversion, names $\widetilde{x}$ can be assumed to be distinct; and if $x \notin \mathsf{fn}(Q) \cup \{\widetilde{x}\}$ then in $(\boldsymbol{\nu} x, \widetilde{x}\,)\langle Q \rangle P$ the restriction on name x can be pushed inwards, resulting in the standard concretion $\boldsymbol{\nu} \widetilde{x}\, \langle Q \rangle (\boldsymbol{\nu} x\, P)$. In the following, we therefore assume that if $x \notin \mathsf{fn}(Q) \cup \{\widetilde{x}\}$, then $(\boldsymbol{\nu} x, \widetilde{x}\,)\langle Q \rangle P$ denotes $\boldsymbol{\nu} \widetilde{x}\, \langle Q \rangle (\boldsymbol{\nu} x\, P)$.

We wish to extend restriction to operate on abstractions, and (a form of) parallel composition to operate on abstractions and concretions:

Let $F = (X)\,Q$:

— if $X \notin \mathsf{fv}(P)$ then $F \mid P$ denotes $(X)\,(Q \mid P)$

Prefix	$\mu.\, A \xrightarrow{\mu} A$		
Parallelism	$P_1 \xrightarrow{\mu} A$	implies	$P_1 \mid P_2 \xrightarrow{\mu} A \mid P_2$
First-order communication	$P_1 \xrightarrow{m} P_1'$		
	$P_2 \xrightarrow{\overline{m}} P_2'$	implies	$P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'$
Higher-order communication	$P_1 \xrightarrow{a} F$		
	$P_2 \xrightarrow{\overline{a}} C$	implies	$P_1 \mid P_2 \xrightarrow{\tau} F \bullet C$
Restriction	$P \xrightarrow{\mu} A,\ \mu \notin \{x, \overline{x}\}$	implies	$\boldsymbol{\nu} x\, P \xrightarrow{\mu} \boldsymbol{\nu} x\, A$
Constants	$F \circ P \xrightarrow{\mu} A$		
	$K \stackrel{\triangle}{=} F$	implies	$K \circ P \xrightarrow{\mu} A$

Table 1.1: The transition system

(and similarly for $P \mid F$),

— $\boldsymbol{\nu} x\, F$ denotes $(X)\, \boldsymbol{\nu} x\, Q$;

if $C = \boldsymbol{\nu}\widetilde{x}\, \langle Q \rangle R$ then

if $\widetilde{x} \cap \mathsf{fn}(P) = \emptyset$ then $C \mid P$ denotes $\boldsymbol{\nu}\widetilde{x}\, \langle Q \rangle (R \mid P)$

(and similarly for $P \mid C$).

We now present the operational semantics of the calculus. First, we define an operation of pseudo-application between an abstraction $F = (X)\, P$ and a concretion $C = \boldsymbol{\nu}\widetilde{x}\, \langle Q \rangle R$. By alpha conversion, we can assume that $\widetilde{x} \cap \mathsf{fn}(F) = \emptyset$, and then we set

$$C \bullet F \stackrel{\text{def}}{=} \boldsymbol{\nu}\widetilde{x}\, (R \mid P\{{}^{Q}/_{X}\})$$

and, symmetrically,

$$F \bullet C \stackrel{\text{def}}{=} \boldsymbol{\nu}\widetilde{x}\, (P\{{}^{Q}/_{X}\} \mid R)\,.$$

The operational semantics of the calculus is reported in Table 1.1. We have omitted the symmetric forms of the parallelism and communication rules. The following forms of

judgements are introduced:

$$\begin{array}{ll} P \xrightarrow{a} F & \text{(higher-order input transition at port } a\text{)} \\ P \xrightarrow{\overline{a}} C & \text{(higher-order output transition at port } a\text{)} \\ P \xrightarrow{\ell} Q & \text{(first-order transition)} \end{array}$$

where P, Q, F, C are agents. In turn, a first-order transition can be a first-order input (if $\ell \in \mathcal{F}$), a first-order output (if $\ell \in \overline{\mathcal{F}}$), or an interaction (if $\ell = \tau$).

We shall normally put enough brackets in the expressions so to avoid precedence ambiguities among the operators. However, to reduce the number of brackets, in a few places we shall assume the following syntactic rules: substitutions and notations "$\bullet$" and "$\circ$" have the highest syntactic precedence; the abstraction and concretion constructs the lowest; parallel composition has weaker precedence than the other process constructs. For instance, $\langle P \rangle !m.\, R \mid Q$ stands for $\langle P \rangle ((!m.\, R) \mid Q)$, and $F \bullet C \mid Q\{^R/_X\}$ stands for $(F \bullet C) \mid (Q\{^R/_X\})$.

1.4.2.2 Normal bisimulation

An arguably natural notion of bisimulation for higher-order processes is *context bisimulation*; it exploits the duality between abstractions and concretions, so to test an abstraction with all possible concretions it can be used with, and conversely. Its bisimulation clauses on higher-order inputs and outputs are therefore as follows, if $\mathcal{R}$ is a context-bisimulation candidate:

1. whenever $P \xrightarrow{a} F$, there exists G s.t. $Q \stackrel{a}{\Rightarrow} G$ and $C \bullet F \;\mathcal{R}\; C \bullet G$, for all closed concretions C;

2. whenever $P \xrightarrow{\overline{a}} C$, there exists D s.t. $Q \stackrel{\overline{a}}{\Rightarrow} D$ and $F \bullet C \;\mathcal{R}\; F \bullet D$, for all closed abstractions F.

Surprisingly, the universal quantifications on concretions and abstractions supplied by the external observer may be removed using a special kind of concretion and of abstraction, namely the concretion $\langle \overline{m}.\, \mathbf{0} \rangle \mathbf{0}$ and the abstraction $(X)\, !m.\, X$, where m is supposed to be a name fresh in the tested processes. In the former concretion, the continuation is null and therefore we can simply use the process $\overline{m}.\, \mathbf{0}$ (or even just $\overline{m}$), called a *trigger* and abbreviated $\mathtt{Tr}_m$. Similarly we write $\mathtt{Ab}_m$ as an abbreviation for $(X)\, !m.\, X$. We thus obtain *normal bisimulation*.

Definition 1.4.13 (normal bisimulation)**.** A relation $\mathcal{R} \subseteq \mathrm{Pr} \times \mathrm{Pr}$ is a *normal simulation* if $P \;\mathcal{R}\; Q$ implies, for $m \notin fn(P, Q)$:

1. whenever $P \xrightarrow{\ell} P'$, there exists Q' s.t. $Q \stackrel{\hat{\ell}}{\Rightarrow} Q'$ and $P' \;\mathcal{R}\; Q'$;

2. whenever $P \xrightarrow{a} F$, there exists G s.t. $Q \stackrel{a}{\Rightarrow} G$ and $F \circ \mathtt{Tr}_m \;\mathcal{R}\; G \circ \mathtt{Tr}_m$;

3. whenever $P \xrightarrow{\overline{a}} C$, there exists D s.t. $Q \stackrel{\overline{a}}{\Rightarrow} D$ and $C \bullet \mathtt{Ab}_m \;\mathcal{R}\; D \bullet \mathtt{Ab}_m$.

A relation $\mathcal{R}$ is a *normal bisimulation*, in symbols $\approx$-bisimulation, if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are normal simulations. We say that P and Q are *normal bisimilar*, in symbols $P \approx Q$, if $P\ \mathcal{R}\ Q$, for some $\approx$-bisimulation $\mathcal{R}$.

In clauses (2) and (3) of Definition 1.4.13 it is enough to pick *some* fresh name m, since the specific choice of the fresh name does not affect the equivalence of the resulting processes. The extension of $\approx$ to closed abstractions and open agents is defined similarly, only employing fresh triggers. For closed concretions C and D we set $C \approx D$ if $C \bullet \mathtt{Ab}_m \approx D \bullet \mathtt{Ab}_m$, for some fresh m.

Theorem 1.4.14. *Normal bisimilarity is a congruence relation in HOπ.*

The proof in [San96a] derives the congruence of normal bisimilarity from congruence of context bisimilarity. In doing so, an auxiliary relation, triggered bisimilarity, is used for an intermediate characterisation. We are not aware of a simpler and direct proof of congruence for normal bisimulation, using the bisimulation proof method. The hard case is that of parallel composition. Usually, substitutivity of a bisimilarity with respect to an operator is established using, as a candidate relation, the closure of the bisimilarity under the operator itself (possibly enhanced with an auxiliary operator such as restriction), and using some 'bisimulation up-to' techniques to simplify the reasoning.

For normal bisimilarity and parallel composition, we have to prove that $P \approx Q$ implies $P \mid T \approx Q \mid T$ for all T. In the case of a higher-order interaction between P and T, the hypotheses at hand can be used to show the existence of a matching interaction between Q and T, but this is not sufficient to conclude. Normal bisimilarity guarantees that P and Q remain related after a higher-order input only when a trigger process is received; however T may transmit an arbitrary process. Replacing a trigger with an arbitrary process is possible as an algebraic law. However the replacement is unsound within up-to techniques. The natural up-to technique would be the 'up-to expansion' technique (*expansion* is a behavioural preorder, finer than weak bisimilarity, capturing an idea of 'process efficiency'). For the technique to be sound, however, the expansion preorder must be applied in a specific direction, whereas the algebraic transformation needed for the replacement of a trigger would require the opposite direction. Similar problems exist on interactions between P and T, and between Q and T when P and Q perform higher-order outputs.

1.4.2.3 Equations in HOπ

We use bold letters $\mathbf{X}, \mathbf{Y}, \ldots$ for *equation variables*. The body of an equation is an abstraction possibly containing equation variables. We stick to monadic abstractions, as in the previous sections.

We call *extended HOπ* the HOπ syntax of Section 1.4.2.1 extended with equation variables. To make the distinction between the ordinary syntax of HOπ and the syntax of extended HOπ, we use bold letters to range over the syntactic categories of the latter. Thus, in extended HOπ, $\mathbf{P}, \mathbf{Q}$ stand for *extended processes*, $\mathbf{F}$ for *extended abstractions*, $\mathbf{C}$ for *extended concretions*, and $\mathbf{A}$ for extended agents (the word 'extended' is used here to denote the fact that equation variables may occur). The grammar for $\mathbf{F}$ is:

$$\mathbf{F} \quad ::= \quad (X)\,\mathbf{P} \;\Big|\; K \;\Big|\; \mathbf{X}.$$

Extended agents can be either extended processes, extended concretions or extended abstractions.

Example 1.4.15. We report some examples of systems of equations, whose solutions are then discussed in Example 1.4.17:

1. $\mathbf{X} = (Y)\,(\mathbf{X} \circ Y)$.
2. $\mathbf{X} = (Y)\,(\mathbf{X} \circ Y \mid Y)$.
3. $\mathbf{X} = (X)\,a.\,(Y)(X \mid Y)$ (where the variable $\mathbf{X}$ of the equation does not occur in the body) .
4. $\begin{aligned} \mathbf{X} &= (X)\,d.\,(Z)\,\mathbf{Y} \circ Z \\ \mathbf{Y} &= (Z)\,r.\,(Z \mid \mathbf{Y} \circ Z) \end{aligned}$

If the parameter of the equation is not important, we omit it, as in the equation $\mathbf{X} = a.\,(X)\overline{b}.\langle X\rangle\mathbf{X}$, whose solution is a link process that repeatedly receives a process at a and emits it at b.

We write $\mathbf{A}[\widetilde{F}]$ for the expression resulting from $\mathbf{A}$ by replacing each variable $\mathbf{X}_i$ with the abstraction F_i, assuming $\widetilde{F}$ and $\widetilde{\mathbf{X}}$ have the same length.

Remark 1.4.16. *To avoid issues with restriction, we take the replacement in $\mathbf{A}[\widetilde{F}]$ to be a substitution, not a syntactic replacement (i.e., in the substitution, free names of an F_i may not be bound by a restriction in $\mathbf{A}$). An alternative would have been to impose abstractions without free names; this would however introduce issues related to name mobility and make the proofs of the main results more complex.*

As for the plain syntax, so in the extended syntax we assume a normalisation of application redexes; thus any subterm of the form $((X)\,P)\circ Q$ should be thought of as an abbreviation for $P\{^{Q}/_{X}\}$.

Example 1.4.17. Consider the equations from Example 1.4.15:

1. the equation $\mathbf{X} = (Y)\,(\mathbf{X} \circ Y)$ is satisfied by any closed abstraction of the form $(Y)\,P$.
2. In $\mathbf{X} = (Y)\,(\mathbf{X} \circ Y \mid Y)$, the abstraction $(Y)\,(P \mid !Y)$ is solution, for any P.
3. The body of the equation $\mathbf{X} = (X)\,a.\,(Y)(X \mid Y)$ does not use the variable $\mathbf{X}$, hence the only solution is precisely its defining abstraction $(X)\,a.\,(Y)(X \mid Y)$.
4. The system of equations

$$\begin{aligned} \mathbf{X} &= (X)\,d.\,(Z)\,\mathbf{Y} \circ Z \\ \mathbf{Y} &= (Z)\,r.\,(Z \mid \mathbf{Y} \circ Z) \end{aligned}$$

has a unique solution. The abstraction for $\mathbf{X}$ receives a process at d, and then makes it available at r, so that any output at r will start a copy of the received process.

Example 1.4.18. We show the syntactic solutions of some of the equations in Example 1.4.17.

1. The syntactic solution of the equation $\mathbf{X} = (Y)\,\mathbf{X} \circ Y$ is the constant $K \stackrel{\triangle}{=} (Y)\,(K \circ Y)$. This recursive process actually behaves as $\mathbf{0}$.

2. The syntactic solution of the equation $\mathbf{X} = (Y)\,(\mathbf{X} \circ Y \mid Y)$ is the constant $K \stackrel{\triangle}{=} (Y)\,(K \circ Y \mid Y)$, which receives a process and runs infinitely many copies of it.

3. The syntactic solution of the equation $\mathbf{X} = (X)\,a.\,(Y)(X \mid Y)$ is the constant $K \stackrel{\triangle}{=} (X)\,a.\,(Y)(X \mid Y)$ — as the equation does not use equation variables, the expression of the equation is the same as the expression defining the constant.

The lemma below shows a form of commutativity on the order in which replacements are made, when unfolding equations and instantiating its variables.

Lemma 1.4.19. *For any extended agent* $\mathbf{A}$*, and extended abstractions* $\widetilde{\mathbf{F}}$ *and* $\widetilde{\mathbf{F}}'$*, we have*

$$\mathbf{A}[\widetilde{\mathbf{F}}][\widetilde{\mathbf{F}}'] = \mathbf{A}[\widetilde{\mathbf{F}}[\mathbf{F}']] \ .$$

Note that in the above lemma, $\widetilde{\mathbf{F}}$ and $\widetilde{\mathbf{F}}'$ need not have the same length. A special case of the above lemma is when $\widetilde{\mathbf{F}}'$ are simple (i.e., non extended) abstractions, say $\widetilde{F'}$. In this case we obtain $\mathbf{A}[\widetilde{\mathbf{F}}][\widetilde{F'}] = \mathbf{A}[\widetilde{\mathbf{F}}[\widetilde{F'}]]$. This observation also holds for other results below, which are often specialised with simple abstractions.

Lemma 1.4.20. *For any extended agent* $\mathbf{A}$*, we have that* $\mathbf{A}[\widetilde{\mathbf{F}}]$ *is guarded if either* $\mathbf{A}$ *is guarded, or all extended abstractions* $\widetilde{\mathbf{F}}$ *are guarded.*

We use the SOS rules of Table 1.1 also on the syntax of extended HOπ. The following lemma relates transitions of terms of extended HOπ, possibly containing free equation variables, with terms resulting from instantiations of such variables.

Lemma 1.4.21.

1. *Given* $\mathbf{P}$ *an extended process and* $\mathbf{A}$ *an extended agent, if* $\mathbf{P} \xrightarrow{\mu} \mathbf{A}$*, then* $\mathbf{P}[\widetilde{\mathbf{F}}] \xrightarrow{\mu} \mathbf{A}[\widetilde{\mathbf{F}}]$*, for all extended abstractions* $\widetilde{\mathbf{F}}$*.*

2. *If* $\mathbf{P}$ *is a guarded extended process and* $\mathbf{P}[\widetilde{\mathbf{F}}] \xrightarrow{\mu} \mathbf{A}_0$*, then there is an extended agent* $\mathbf{A}$ *such that* $\mathbf{P} \xrightarrow{\mu} \mathbf{A}$ *and* $\mathbf{A}_0 = \mathbf{A}[\widetilde{\mathbf{F}}]$*.*

Proof. Both cases are treated by induction on the derivation of the transition ($\mathbf{P} \xrightarrow{\mu} \mathbf{A}$ or $\mathbf{P}[\widetilde{\mathbf{F}}] \xrightarrow{\mu} \mathbf{A}_0$).

□

Both properties of the previous lemma are valid, as a special case, for transitions emanating from $\mathbf{P}[\widetilde{F}]$.

The following lemma states basic properties about $\circ$ and $\bullet$; it is used implicitly in several places below.

Lemma 1.4.22.

- *For all* $\mathbf{F}$, $\widetilde{F}$, *and* Q, *we have* $(\mathbf{F} \circ Q)[\widetilde{F}] = \mathbf{F}[\widetilde{F}] \circ Q$.
- *For all* $\mathbf{C}$, $\widetilde{F}$ *and* F, *we have* $(\mathbf{C} \bullet F)[\widetilde{F}] = \mathbf{C}[\widetilde{F}] \bullet F$.

Recall that for a name m, a *trigger* at m is a process $\mathtt{Tr}_m \stackrel{\text{def}}{=} \overline{m}.\,\mathbf{0}$, often simply written $\overline{m}$, and that $\mathtt{Ab}_m \stackrel{\text{def}}{=} (X)!m.\,X$. Thus, given an extended concretion $\mathbf{C} = \langle \mathbf{P}_1 \rangle \mathbf{P}_2$ (resp. an extended abstraction $\mathbf{F} = (X)\,\mathbf{P}$), we have $\mathbf{C} \bullet \mathtt{Ab}_m = !m.\,\mathbf{P}_1 \mid \mathbf{P}_2$ (resp. $\mathbf{F} \circ \mathtt{Tr}_m = \mathbf{P}\{\overline{m}/X\}$).

1.4.2.4 Unique solution in HOπ

Divergences in HOπ. In HOπ, a divergence in a process consists of a finite sequence of transitions (of any kind) followed by an infinite sequence of τ transitions. In other words, before embarking in a sequence of internal moves, a diverging process may perform some higher-order interactions with its environment. In order to account for these, we follow the definition of normal bisimulation (Definition 1.4.13), and instantiate agents with the special forms for triggers and abstractions (processes $\mathtt{Tr}_m$ and $\mathtt{Ab}_m$). By using such processes with freshly generated names, we avoid possible interferences with other transitions of the process, and do not break divergences.

We first introduce the notion of reduct, which then allows us to define divergences.

Definition 1.4.23 (Reducts)**.**

1. We say that $\mathbf{P}$ *reduces to* $\mathbf{P}'$, written $\mathbf{P} \rightarrowtriangle \mathbf{P}'$, if one of the following holds:
   (a) $\mathbf{P} \stackrel{\ell}{\rightarrow} \mathbf{P}'$;
   (b) $\mathbf{P}' = \mathbf{C} \bullet \mathtt{Ab}_m$ for some fresh m and some $\mathbf{C}$ such that $\mathbf{P} \stackrel{\overline{a}}{\rightarrow} \mathbf{C}$;
   (c) $\mathbf{P}' = \mathbf{F} \circ \mathtt{Tr}_m$ for some fresh m and some $\mathbf{F}$ such that $\mathbf{P} \stackrel{a}{\rightarrow} \mathbf{F}$.

   Relation $P \rightarrowtriangle P'$ ("P reduces to P'") is defined in the same way for processes that do not contain equation variables.
2. The *set of reducts* of an extended process $\mathbf{P}$, written $\mathtt{red}(\mathbf{P})$, is given by:

$$\mathtt{red}(\mathbf{P}) \stackrel{\triangle}{=} \bigcup_n \{\mathbf{P}_n \mid \mathbf{P} \rightarrowtriangle \mathbf{P}_1 \ldots \rightarrowtriangle \mathbf{P}_n \text{ for some } n \text{ and } \mathbf{P}_i\ (1 \leq i \leq n)\ \}.$$

   Again, the *set of reducts* of a process P, $\mathtt{red}(P)$, is defined in the same way.

3. The set of *reducts of the unfoldings* of a system of equations $\{\mathbf{X}_i = \mathbf{F}_i\}_{i\in I}$, written $\mathtt{red}_\omega(\widetilde{\mathbf{F}})$, is defined as

$$\mathtt{red}_\omega(\widetilde{\mathbf{F}}) \quad \triangleq \quad \bigcup_{n\in\mathbb{N}, i\in I, m \text{ fresh}} \mathtt{red}(\mathbf{F}_i^n \circ \mathtt{Tr}_m).$$

Theorem 1.4.24 (Unique solution)**.** *A guarded system of equations whose syntactic solutions do not diverge has a unique solution for* $\approx$.

Theorem 1.4.25 (Unique solution with innocuous divergences)**.** *Let* $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$ *be a system of guarded equations, and* $\widetilde{K}_{\widetilde{\mathbf{F}}}$ *be its syntactic solutions. If for any* i*, all divergences of* $K_{\widetilde{\mathbf{F}},i}$ *are innocuous, then* $\widetilde{\mathbf{F}}$ *has a unique solution for* $\approx$.

An example of an innocuous divergence is when the body of an equation $\mathbf{P}$ is able to output a certain process P. In this case, the syntactic solution of $\mathbf{P}$ inherits the divergences of P, which are innocuous.

Example 1.4.26. Consider the equation

$$\mathbf{X} = \overline{a}.\langle P\rangle\mathbf{P}$$

where P diverges (take for instance $P = !\tau$). The syntactic solution of this equation diverges: indeed we have (using $\twoheadrightarrow$ from Definition 1.4.23):

$$\overline{a}.\langle P\rangle\mathbf{P} \twoheadrightarrow !m.\, P \mid \mathbf{P} \ .$$

After an input on m, P is active, and the divergence can happen. However this divergence is innocuous, and does not prevent unique solution, as there is no need to unfold the equation more than once to unleash the divergence.

We now show an example of a non-innocuous divergence that does not prevent the equation from having a unique solution.

Example 1.4.27. Consider the single equation

$$\mathbf{X} = \overline{a}.\langle !\,\overline{n}\rangle \ \mid \ n.\,\mathbf{X} \ .$$

Its syntactic solution, which is given by $K \triangleq \overline{a}.\langle !\,\overline{n}\rangle \mid n.\,K$, has a divergence. Indeed we have (again, using $\twoheadrightarrow$ from Definition 1.4.23):

$$\begin{array}{rcl} K & \twoheadrightarrow & !m.\,!\overline{n} \mid n.\,K \quad \text{for } m \text{ fresh (i.e., } m \neq n) \\ & \twoheadrightarrow & !m.\,!\overline{n} \mid !\overline{n} \mid n.\,K \end{array}$$

and the latter process can do an infinite sequence of τ-transitions, through synchronisations on channel n. This divergence involves the unfolding of K after each τ-step, so it is not innocuous.

However, the equation has a unique solution, intuitively because the transition at $\overline{a}$, which unleashes the divergence, can be avoided; therefore such a divergence causes no harm. We see here, therefore, a limitation of the notion of innocuous divergence.

Remark 1.4.28. We have focused on the monadic calculus and on monadic abstractions, and considered only process passing. The extension to polyadicity and to the communication of abstractions (i.e., parametrised processes) of arbitrarily high type, as in the full HOπ [San93a], appears to be only notationally more complex. However, handling more sophisticated types for the terms exchanged (i.e., abstractions with recursive types or polymorphism) would seem challenging.

1.4.3 A proof with the unique-solution technique

We now discuss another example of equality proof in HOπ, using the unique solution technique.

Example 1.4.29. We consider the proof of equality between the terms $P_1 \stackrel{\text{def}}{=} \overline{a}\,.\,\langle b\,.\,(Y)\,P\rangle$ and $Q_1 \stackrel{\text{def}}{=} \overline{a}\,.\,\langle b\,.\,(Y)\,Q\rangle$, where

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & \boldsymbol{\nu} a\,(d.\,R \mid \overline{a}\,.\,\langle R\rangle) \\ R & \stackrel{\text{def}}{=} & a\,.\,(X)\,(Y \mid d.\,X \mid \overline{a}\,.\,\langle X\rangle) \end{array}$$

and

$$\begin{array}{rcl} Q & \stackrel{\text{def}}{=} & \boldsymbol{\nu} a\,(S \mid \overline{a}\,.\,\langle Y \mid S\rangle) \\ S & \stackrel{\text{def}}{=} & a\,.\,(X)\,(d.\,X \mid \overline{a}\,.\,\langle X\rangle) \end{array}$$

Terms P_1 and Q_1 send on a terms that can receive a process at b and then make this process freely available, at a channel d (i.e., arbitrarily many copies of the process may be activated, using d). Essentially, processes P and Q correspond to two ways of modelling a replication operator in HOπ, with a different internal structure. Indeed, if T is the process received at b, and that replaces Y in P and Q, with the abbreviations $P_T \stackrel{\text{def}}{=} P\{^T\!/\!_Y\}$ and $R_T \stackrel{\text{def}}{=} R\{^T\!/\!_Y\}$, we have

$$\begin{array}{rcl} P_T & \stackrel{d}{\longrightarrow} & \boldsymbol{\nu} a\,(R_T \mid \overline{a}\,.\,\langle R_T\rangle) \\ & \stackrel{\tau}{\longrightarrow} & \boldsymbol{\nu} a\,(T \mid d.\,R_T \mid \overline{a}\,.\,\langle R_T\rangle) \\ & \equiv & T \mid P_T \\ & \stackrel{d}{\longrightarrow} & \ldots \end{array}$$

where $\equiv$ indicates the application of a simple algebraic law for shrinking the scope of a restriction. Similarly, for $Q_T \stackrel{\text{def}}{=} Q\{^T\!/\!_Y\}$ and $S_T \stackrel{\text{def}}{=} S\{^T\!/\!_Y\}$ we have

$$\begin{array}{rcl} Q_T & \stackrel{\tau}{\longrightarrow} & \boldsymbol{\nu} a\,(d.\,(T \mid S_T) \mid \overline{a}\,.\,\langle T \mid S_T\rangle) \\ & \stackrel{d}{\longrightarrow} & \boldsymbol{\nu} a\,(T \mid S_T \mid \overline{a}\,.\,\langle T \mid S_T\rangle) \\ & \equiv & T \mid \boldsymbol{\nu} a\,(S_T \mid \overline{a}\,.\,\langle T \mid S_T\rangle) \\ & \stackrel{\tau}{\longrightarrow} & T \mid \boldsymbol{\nu} a\,(d.\,(T \mid S_T) \mid \overline{a}\,.\,\langle T \mid S_T\rangle) \\ & \stackrel{d}{\longrightarrow} & \ldots \end{array}$$

We prove $P_1 \approx Q_1$ using the technique of unique solution of equations. We use the following equations

$$\begin{array}{rcl} \mathbf{X}_1 & = & \overline{a}\,.\,\langle b\,.\,(Y)\,(\mathbf{X}_2 \circ Y)\rangle \\ \mathbf{X}_2 & = & (Y)\,d.\,(Y \mid (\mathbf{X}_2 \circ Y)) \end{array}$$

The equations are guarded and their unfoldings do not introduce divergences, hence we can apply Theorem 1.4.24. We derive $P_1 \approx Q_1$ by showing that both the pair P_1, $(Y)\,P$ and the pair Q_1, $(Y)\,Q$ are solutions of the system. For the first pair we have to prove

$$\begin{array}{rcl} P_1 & \approx & \overline{a}\,.\,\langle b\,.\,(Y)\,P\rangle \\ (Y)\,P & \approx & (Y)\,d.\,(Y \mid P) \end{array}$$

The first equivalence is trivial, as the two processes are identical. For the second, by definition of normal bisimulation, for some m fresh, and writing P_m as abbreviation for $P\{\overline{m}.\,\mathbf{0}/Y\}$, we have to show

$$P_m \approx d.\,(\overline{m}.\,\mathbf{0} \mid P_m)$$

We have, for $R_m \stackrel{\text{def}}{=} R\{\overline{m}.\,\mathbf{0}/Y\}$:

$$\begin{array}{rcl} P_m & = & \boldsymbol{\nu} a\,(d.\,R_m \mid \overline{a}\,.\,\langle R_m\rangle) \\ & \approx & d.\,\boldsymbol{\nu} a\,(R_m \mid \overline{a}\,.\,\langle R_m\rangle) \\ & = & d.\,\boldsymbol{\nu} a\,(a\,.\,(X)\,(\overline{m}.\,\mathbf{0} \mid d.\,X \mid \overline{a}\,.\,\langle X\rangle) \mid \overline{a}\,.\,\langle R_m\rangle) \\ & \approx & d.\,\boldsymbol{\nu} a\,((\overline{m}.\,\mathbf{0} \mid d.\,R_m \mid \overline{a}\,.\,\langle R_m\rangle)) \\ & \approx & d.\,(\overline{m}.\,\mathbf{0} \mid \boldsymbol{\nu} a\,((d.\,R_m \mid \overline{a}\,.\,\langle R_m\rangle))) \\ & = & d.\,(\overline{m}.\,\mathbf{0} \mid P_m) \end{array}$$

where the uses of $\approx$ are derived using some simple algebraic laws for prefix and restriction plus (for the second occurrence of $\approx$) the law

$$\begin{array}{rcl} \boldsymbol{\nu} a\,(a\,.\,(X)\,T_1 \mid \overline{a}\,.\,\langle T_2\rangle T_3) & \approx & \boldsymbol{\nu} a\,(((X)\,T_1) \bullet (\langle T_2\rangle T_3)) \\ & = & \boldsymbol{\nu} a\,(T_1\{T_2/X\} \mid T_3) \end{array}$$

which is straightforward too, as the process on the right is the only immediate derivative of the process on the left. The reasoning for the other pair, Q_1, $(Y)\,Q$, is similar (in fact, simpler).

The proof above makes use of two equations only. One may find this surprising, because the calculus is higher-order and therefore process terms are exchanged with the environment (i.e., the input at b and the output at a), and because of the recursive structure of the terms compared (their 'fixed-point-like' behaviour). Such reduced size is due both to the choice of normal bisimulation (that does not make use of universal quantifications on concretions or abstractions in the input and output clauses), and to the 'up-to context' flavour of the unique-solution technique.

When comparing the technique of unique solution of equations to the bisimulation proof method [San15] the size of a system of equations is the number of equations involved, whereas the size of a bisimulation is the number of its pairs. Thus the proof above, involving two

equations, would correspond to a bisimulation candidate with two pairs. The proof of a similar result in [SKS11] (Section 6.7, second example) uses 'environmental bisimulation up-to context'. The relation employed in that proof is infinite, because in environmental bisimulation one has to take into account all possible processes that may be received in a higher-order input and because of the limitations of the known forms of 'up-to context' for environmental bisimulation (only in certain clauses of the bisimulation a context may be erased and even in these cases there are syntactic constraints on the context itself).

Chapter 2

Unique solution for Full Abstraction

2.1 Lazy functions as mobile processes

2.1.1 The lazy λ-calculus

Before studying Milner's call-by-value encoding, to serve as a gentle introduction, we revisit the proof of full abstraction for Milner's call-by-name encoding, more specifically the proof of completeness. We skim over already-established result (such as soundness of the encoding, soundness of β-reduction, etc), and over some technical details, to focus on the unique-solution proof itself. The proof for call-by-value is harder, but has the same high-level structure.

We first recall some basic definitions regarding the λ-calculus. We use M, N to range over the set Λ of λ-terms, and x, y, z to range over λ variables.

We assume the standard concepts of free and bound variables and substitutions, and identify α-convertible terms. The terms in Λ are sometimes call *open* to distinguish them from the subset of *closed* terms – those without free variables.

We let x and y range over the set of λ-calculus variables. The set Λ of (open) λ-terms is defined by the grammar

$$M := \; x \;\Big|\; \lambda x.\, M \;\Big|\; M_1 M_2 \,.$$

Free variables, closed terms, substitution, α-conversion etc. are defined as usual [Bar84, HS86]. The "Barendregt convention" allows us to assume freshness of bound variables and names whenever needed. The set of free variables in the term M is written $\mathsf{fv}(M)$. We group brackets on the left; therefore MNL is $(MN)L$. The λ-abstraction $\lambda x. \ldots$ has higher priority, and thus reaches as far as possible to the right. We abbreviate $\lambda x_1. \cdots . \lambda x_n. M$ as $\lambda x_1 \cdots x_n. M$, or $\lambda \widetilde{x}. M$ if the length of $\widetilde{x}$ is not important. Symbol Ω stands for the always-divergent term $(\lambda x.\, xx)(\lambda x.\, xx)$.

As previously, a *context* is a term with a hole $[\cdot]$, possibly occurring more than once. If C is a context, $C[M]$ is a shorthand for C where the hole $[\cdot]$ is substituted by M.

The rules defining the call-by-name strategy, defined on open terms, are the following:

$$(\lambda x.\, M)N \to M\{N/x\} \qquad \frac{M \to M'}{MN \to M'N}$$

As usual $\Rightarrow$ is the reflexive and transitive closure of the single-step reduction $\to$.

We write $M \Uparrow$ if M diverges. If M is an open λ-term, then either M diverges, or $M \Rightarrow \lambda x.\, M'$ (for some x and M'), or $M \Rightarrow xM_1 \dots M_n$ (for some x, M_1, …, M_n).

The following notion of tree is used to provide a semantics for the call-by-name λ-calculus.

Definition 2.1.1 (Lévy-Longo Tree)**.** The *Lévy-Longo Tree* (LT) of an open λ-term M, written $LT(M)$, is the (possibly infinite) tree defined coinductively as follows.

1. If M diverges, then $LT(M)$ is the tree with a single node labelled $\bot$.
2. If $M \Rightarrow \lambda x.\, M'$, then $LT(M)$ is the tree with a root labelled with "$\lambda x.$", and $LT(M')$ as its unique descendant.
3. If $M \Rightarrow xM_1 \dots M_n$, then $LT(M)$ is the tree with a root labelled with "x", and $LT(M_1), \dots, LT(M_n)$ (in this order) as its n descendants.

LT equality (whereby two λ-terms are identified if their LTs are equal) can also be presented as a bisimilarity (*open bisimilarity*, $\simeq^o$), defined as the largest *open bisimulation*.

Definition 2.1.2. A relation $\mathcal{R}$ on Λ is an *open bisimulation* if, whenever $M\ \mathcal{R}\ N$:

1. $M \Rightarrow \lambda x.\, M'$ implies $N \Rightarrow \lambda x.\, N'$ with $M'\ \mathcal{R}\ N'$;
2. $M \Rightarrow xM_1 \dots M_n$ with $n \geq 0$ implies $N \Rightarrow xN_1 \dots N_n$ and $M_i\ \mathcal{R}\ N_i$ for all $1 \leq i \leq n$.
3. The converse of clauses 1 and 2 on the challenges from N.

Theorem 2.1.3 ([San00, SX14])**.** *Let M and N be two λ-terms; then $LT(M) = LT(N)$ iff $M \simeq^o N$.*

2.1.2 Milner's encoding

Milner's encoding of the call-by-name λ-calculus into Aπ [Mil92] is defined in figure 2.1.

Function application is translated into a particular form of parallel combination of two agents, the function and its argument; β-reduction is then modeled as process interaction. Since the syntax of the π-calculus only allows for the transmission of names along channels, the communication of a term is simulated by the communication of a *trigger* for it. The translation of a λ-term is an abstraction that is parametric on a name, the *location* of the λ-term, which is intuitively the name along which the term, as a function, will receive its argument. Precisely, the encoding of a term receives two names along its location p: the

$$[\![\lambda x.\, M]\!] \stackrel{\triangle}{=} (p)\ p(x,q).\, [\![M]\!]\langle q\rangle \qquad\qquad [\![x]\!] \stackrel{\triangle}{=} (p)\ \overline{x}\langle p\rangle$$

$$[\![MN]\!] \stackrel{\triangle}{=} (p)\ \boldsymbol{\nu} r, x\, ([\![M]\!]\langle r\rangle \mid \overline{r}\langle x, p\rangle \mid\, !x(q).\, [\![N]\!]\langle q\rangle)$$

Figure 2.1: Milner's encoding of the call-by-name λ-calculus into Aπ.

first is a trigger for its argument and the second is the location to be used for the next interaction.

The full abstraction theorem for the encoding [San00, SX14] states that two λ-terms have the same LT iff their encodings into Aπ are weakly bisimilar terms. Full abstraction has two components: soundness, which says that if the encodings are weakly bisimilar then the original terms have the same LT; and completeness, which is the converse direction. The proof [San00, SX14] first establishes an operational correspondence between the behaviour (visible and silent actions) of λ-terms and of their encodings. Then, exploiting this correspondence, soundness and completeness are proved using the bisimulation proof method. For soundness, this amounts to following the defining clauses of open bisimulation (Definition 2.1.2). In contrast, completeness involves enhancements of the bisimulation proof method, notably 'bisimulation up to context and expansion'. In the latter, *expansion* is an auxiliary preorder relation, finer than weak bisimilarity. As a consequence, the technique requires having developed the basic theory for the expansion preorder (e.g., precongruence properties and basic algebraic laws), and requires an operational correspondence fine enough in order to be able to reason about expansion (expansion appears within the statements of operational correspondence).

On the other hand, soundness is mostly technical, and does not require as advanced proof techniques.

2.1.3 Completeness with the unique solution technique

We show that, by appealing to unique solution of equations, completeness can be proved by defining an appropriate system of equations, each equation having a simple shape, and without the need for auxiliary preorders.

Prerequisite. For this, the only results needed are: (*i*) validity of β-reduction for the encoding (Lemma 2.1.4), whose proof is simple and consists in the application of a few algebraic laws (including laws for replication); (*ii*) the property that if M diverges then $[\![M]\!]\langle p\rangle$ may never produce a visible action (Lemma 2.1.5); (*iii*) a Lemma for rearranging parallel composition and restrictions in the process encoding $xM_1 \ldots M_n$ (Lemma 2.1.6).

Lemma 2.1.4 (Validity of β-reduction, [San00])**.** *For $M \in \Lambda$, if $M \to M'$ then $[\![M]\!] \approx [\![M']\!]$.*

Lemma 2.1.5. *If M diverges, then $[\![M]\!] \approx (p)\ \mathbf{0}$.*

Lemma 2.1.6 ([San00]). *For any $M_1, \ldots, M_n \in \Lambda$, and variable x, we have*

$$[\![x\ M_1\ \ldots\ M_n]\!] = (p)\ (\boldsymbol{\nu} r_0, \ldots, r_n\,)\big(\overline{x}\langle r_0\rangle \mid \overline{r_0}\langle r_1, x_1\rangle \mid \ldots \mid \overline{r_{n-1}}\langle r_n, x_n\rangle \mid \\ !x_1(q_1).\,[\![M_1]\!]\langle q_1\rangle \mid \ldots \mid !x_n(q_n).\,[\![M_n]\!]\langle q_n\rangle\big)\ .$$

The system of equations. Suppose M_0 and N_0 are two λ-terms with the same LT. We define a system of equations $\mathcal{E}_{\mathcal{R}}$, whose solutions are obtained from the encodings of the Lévy-Longo trees of M_0 and N_0. We will then use Theorem 1.4.7 to deduce $[\![M_0]\!] \approx [\![N_0]\!]$.

Since M_0 and N_0 have the same LT, then by Theorem 2.1.3 there is an open bisimulation $\mathcal{R}$ containing the pair (M_0, N_0). The variables of the equations are of the form $X_{M,N}$ for $M\mathcal{R}N$, and there is one equation for each pair in $\mathcal{R}$.

We consider a pair (M, N) in $\mathcal{R}$, and explain how the corresponding equation is defined. The equation is parametrised on the free variables of M and N (to ensure that its body is a name-closed abstraction) together with an additional continuation name (as all encodings of terms).

We assume an ordering of the λ-calculus variables so to be able to view a finite set of variables as a tuple. This allows us to write $\widetilde{x}$ for the variables appearing free in M or N.

The equations are the translation of the clauses of Definition 2.1.2, assuming a generalisation of the encoding to equation variables by adding the clause: $[\![X_{M,N}]\!] \stackrel{\text{def}}{=} (\widetilde{x}, p)\ X_{M,N}\langle \widetilde{x}, p\rangle$.

Since $M\mathcal{R}N$, then either both M and N diverge, or they satisfy one of the two clauses of Definition 2.1.2.

- If M, N are both divergent, then the equation is

  $$X_{M,N} = (\widetilde{x}, p)\ !\tau$$

  (as $!\tau \approx [\![\Omega]\!]$)

- If M, N satisfy clause 1 of Definition 2.1.2, the equation is

  $$X_{M,N} = (\widetilde{x}, p)\ [\![\lambda x.\, X_{M',N'}]\!]\langle p\rangle$$

  that is,

  $$X_{M,N} = (\widetilde{x}, p)\ p(x, q).\, X_{M',N'}\langle \widetilde{y}, q\rangle$$

  where $\widetilde{y}$ are the free variables in M', N'.

- If M, N satisfy clause 2 of Definition 2.1.2, the equation is given by the translation of $xX_{M_1,N_1} \ldots X_{M_n,N_n}$, which, rearranging restrictions and parallel compositions (Lemma 2.1.6), can be written

  $$X_{M,N} = (\widetilde{x}, p)(\boldsymbol{\nu} r_0, \ldots, r_n\,)\big(\ \overline{x}\langle r_0\rangle \mid \overline{r_0}\langle r_1, x_1\rangle \mid \ldots \mid \overline{r_{n-1}}\langle r_n, x_n\rangle \mid \\ !x_1(q_1).\, X_{M_1,N_1}\langle \widetilde{x_1}, q_1\rangle \mid \ldots \mid !x_n(q_n).\, X_{M_n,N_n}\langle \widetilde{x_n}, q_n\rangle\ \big)$$

  where $\widetilde{x_i}$ are the free variables in M_i, N_i.

Essentially, each equation above represents the translation of a specific node of the LT for M and N.

We need $\mathcal{E}_{\mathcal{R}}$ to have a unique solution.

Lemma 2.1.7. *The system of equations $\mathcal{E}_{\mathcal{R}}$ has a unique solution.*

Proof. We rely on Lemma 1.4.8 to show that the equations may only produce innocuous divergences. It is easy to check that the syntactic condition holds: a location name may only appear once (in input position); a trigger name either appears once (as a replicated input), or it only appears in output position.

As a consequence, since the labelled transition system is ground (names are only replaced by fresh ones), no τ-transition can ever be performed, after any number of visible actions. Further, $\mathcal{E}'_{\mathcal{R}}$ is guarded. Hence we can apply Theorem 1.4.11. □

The solutions. We now define the set of solutions of $\mathcal{E}_{\mathcal{R}}$. For $(M, N) \in \mathcal{R}$, we set $F_{M,N}$ to be the abstraction $(\widetilde{x}, p)\ [\![M]\!]\langle p\rangle$, and similarly $G_{M,N} \triangleq (\widetilde{x}, p)\ [\![N]\!]\langle p\rangle$.

Lemma 2.1.4 allows us to show that the set of all such abstractions $F_{M,N}$ yields a solution for the system of equations, and similarly for $G_{M,N}$.

Lemma 2.1.8. *The sets of abstractions $(F_{M,N})_{M\mathcal{R}N}$ and $(G_{M,N})_{M\mathcal{R}N}$ are solutions of $\mathcal{E}_{\mathcal{R}}$.*

Proof. We reason by cases, following Definition 2.1.2.

- If clause (1) of Definition 2.1.2 holds, then $M \Rightarrow xM_1 \dots M_n$ and we have

$$\begin{aligned} F_{M,N} &= (\widetilde{y}, p)\ [\![M]\!]\langle p\rangle \\ &\approx (\widetilde{y}, p)\ [\![xM_1 \dots M_n]\!]\langle p\rangle && \text{(by Lemma 2.1.4)} \\ &= (\widetilde{y}, p)\ p(x, q).\, [\![M']\!]\langle q\rangle \\ &= (\widetilde{y}, p)\ \big(p(x, q).\, X_{M',N'}\langle\widetilde{z}\rangle\big)\{F_{M',N'}/X_{M',N'}\} \end{aligned}$$

  where $\widetilde{z}$ is a subset of $x, \widetilde{y}$, containing the free variables of M and N.

- If clause (2) holds, then $M \Rightarrow \lambda x.\, M'$ and we have

$$\begin{aligned} F_{M,N} &= (\widetilde{y}, p)\ [\![M]\!]\langle p\rangle \\ &\approx (\widetilde{y}, p)\ (\boldsymbol{\nu} r_0, \dots, r_n\,)\big(\overline{x}\langle r_0\rangle \mid \overline{r_0}\langle r_1, x_1\rangle \mid \dots \mid \overline{r_{n-1}}\langle r_n, x_n\rangle \mid \\ &\qquad !x_1(q_1).\, [\![M_1]\!]q_1 \mid \dots \mid !x_n(q_n).\, [\![M_n]\!]q_n\,\big) && \text{(by Lemmas 2.1.4 and 2.1.6)} \\ &\approx (\widetilde{y}, p)\ (\boldsymbol{\nu} r_0, \dots, r_n\,)\big(\overline{x}\langle r_0\rangle \mid \overline{r_0}\langle r_1, x_1\rangle \mid \dots \mid \overline{r_{n-1}}\langle r_n, x_n\rangle \mid \\ &\qquad !x_1(q_1).\, X_{M_1,N_1}\langle\widetilde{x_1}, q_1\rangle \mid \dots \mid !x_n(q_n).\, X_{M_n,N_n}\langle\widetilde{x_n}, q_n\rangle\,\big) \\ &\qquad \{F_{M_1,N_1}/X_{M_1,N_1}, \dots, F_{M_n,N_n}/X_{M_n,N_n}\} \end{aligned}$$

- If neither clause (1) or (2) hold, meaning both M and N diverge, then we have

$$\begin{aligned} F_{M,N} &= (\widetilde{x}, p)\ [\![M]\!]\langle p\rangle \\ &\approx (\widetilde{x}, p)\ \mathbf{0} && \text{(by Lemma 2.1.5)} \\ &\approx (\widetilde{x}, p)\ !\tau \end{aligned}$$

□

Completeness. We can now show the completeness of the encoding w.r.t. Lévy-Longo trees.

Proposition 2.1.9 (Completeness, [San00])**.** *For $M, N \in \Lambda$, $LT(M) = LT(N)$ implies $[\![M]\!] \approx [\![N]\!]$.*

Proof. By combining Lemmas 2.1.7 and 2.1.8 . We have that $(F_{M,N})_{M\mathcal{R}N}$ and $(G_{M,N})_{M\mathcal{R}N}$ are both solutions of the system of equations, hence they are bisimilar. Finally,

$$(\widetilde{x}, p)\ [\![M]\!]_0\langle p\rangle = F_{M_0,N_0} \approx G_{M_0,N_0} = (\widetilde{x}, p)\ [\![N]\!]_0\langle p\rangle$$

and thus $[\![M_0]\!] \approx [\![N_0]\!]$. □

2.2 Eager functions as mobile processes

2.2.1 Call-by-value reduction semantics

We consider the usual *call-by-value* reduction semantics [Plo75] with a 'weak' paradigm (no reductions occur under an abstraction), as for call-by-name, and adapted to open terms. As discussed in [AG16], if we were to work with a strong reduction strategy (e.g., reductions do happen below a λ-abstraction), working with open terms is unavoidable. Our presentation of call-by-value follows in the steps of Lassen; for more details on the reduction semantics of call-by-value, we refer the reader to [Las05, AG16, RP04].

To define call-by-value's β_v-reduction, we need to enforce that evaluation may only occur in a specific part of the term; this is the role of evaluation contexts. An *evaluation context* is a special kind of inductively defined context, with exactly one hole $[\cdot]$, and in which a term replacing the hole can immediately run. In the pure λ-calculus *values* are abstractions and variables.

$$\begin{array}{rrcl} \text{Evaluation contexts} & C_{\mathrm{e}} & := & [\cdot] \mid C_{\mathrm{e}}M \mid VC_{\mathrm{e}} \\ \text{Values} & V & := & x \mid \lambda x.\, M \end{array}$$

We write $\mathsf{fv}(C_{\mathrm{e}})$ for the free variables of C_{e}.

Eager reduction (or β_{v}-reduction), $\longrightarrow \subseteq \Lambda \times \Lambda$, is defined on open terms, and is determined by the rule:

$$C_{\mathrm{e}}[(\lambda x.\, M)V] \longrightarrow C_{\mathrm{e}}[M\{V/x\}]\ .$$

We write $\Longrightarrow$ for the reflexive transitive closure of $\longrightarrow$. A term in *eager normal form* is a term that has no eager reduction.

Proposition 2.2.1. *The following hold:*

1. *If $M \longrightarrow M'$, then $C_{\mathrm{e}}[M] \longrightarrow C_{\mathrm{e}}[M']$ and $M\sigma \longrightarrow M'\sigma$, for any substitution σ that replaces variables with values.*

2. *Terms in eager normal form are either values or admit a unique decomposition of the shape $C_{\mathrm{e}}[x\ V]$.*

Therefore, given a term M, either $M \Longrightarrow M'$ where M' is a term in eager normal form, or there is an infinite reduction sequence starting from M. In the first case, we say that M *has eager normal form* M', written $M \Downarrow M'$; in the second case, we say that M *diverges*, written $M \Uparrow$. We write $M \Downarrow$ when $M \Downarrow M'$ for some M'.

Definition 2.2.2 (Contextual equivalence)**.** Given $M,\ N \in \Lambda$, we say that M and N are contextually equivalent, written $M \simeq^{\Lambda}_{\mathrm{ct}} N$, if for any context C, we have $C[M] \Downarrow$ iff $C[N] \Downarrow$.

2.2.1.1 Tree semantics for the call-by-value λ-calculus

In this section, we present Lassen's *eager normal-form bisimilarity* [Las05, LL05, SL09].

Definition 2.2.3 (Eager normal-form bisimulation)**.** A relation $\mathcal{R}$ between λ-terms is an *eager normal-form bisimulation* if, whenever $M\ \mathcal{R}\ N$, one of the following holds:

1. both M and N diverge;
2. $M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C'_{\mathrm{e}}[xV']$ for some x, values V, V', and evaluation contexts C_{e} and C'_{e} satisfying $V\ \mathcal{R}\ V'$ and $C_{\mathrm{e}}[z]\ \mathcal{R}\ C'_{\mathrm{e}}[z]$ for a fresh z;
3. $M \Downarrow \lambda x.\, M'$ and $N \Downarrow \lambda x.\, N'$ for some x, M', N' with $M'\ \mathcal{R}\ N'$;
4. $M \Downarrow x$ and $N \Downarrow x$ for some x.

Eager normal-form bisimilarity, $\leftrightarrows$, is the largest eager normal-form bisimulation.

Essentially, the structure of a λ-term that is unveiled by Definition 2.2.3 is that of a (possibly infinite) tree obtained by repeatedly applying β_{v}-reduction, and branching a tree whenever instantiation of a variable is needed to continue the reduction (clause (2)). We call such trees *Eager Trees* (ETs) and accordingly also call eager normal-form bisimilarity the *Eager-Tree equality*.

Example 2.2.4. Relation $\leftrightarrows$ is strictly finer than contextual equivalence $\simeq^{\Lambda}_{\mathrm{ct}}$: the inclusion $\leftrightarrows \subseteq \simeq^{\Lambda}_{\mathrm{ct}}$ follows from the congruence properties of $\leftrightarrows$ [Las05]. For strictness, examples are given by the following equalities, which hold for $\simeq^{\Lambda}_{\mathrm{ct}}$ but not for $\leftrightarrows$:

$$\Omega = (\lambda y.\, \Omega)(xV) \qquad xV = (\lambda y.\, xV)(xV)\ .$$

Example 2.2.5 (η rule)**.** The η-rule is not valid for $\leftrightarrows$. For instance, we have $\Omega \not\leftrightarrows \lambda x.\, \Omega x$. The rule is not even valid on values, as we also have $\lambda y.\, xy \not\leftrightarrows x$. It holds however for abstractions: $\lambda y.\, (\lambda x.\, M)y \leftrightarrows \lambda x.\, M$ when $y \notin \mathsf{fv}(M)$.

The failure of the η-rule $\lambda y.\, xy \not\leftrightarrows x$ is troublesome as, under any closed value substitution (a substitution replacing variables with closed values), the two terms are indeed eager normal-form bisimilar. Thus *η-eager normal-form bisimilarity* [Las05] takes η-expansion into account so to recover such missing equalities.

Definition 2.2.6 (η-eager normal-form bisimulation)**.** A relation $\mathcal{R}$ between λ-terms is an *η-eager normal-form bisimulation* if, whenever $M\ \mathcal{R}\ N$, either one of the clauses of Definition 2.2.3, or one of the two following additional clauses, hold:

5. $M \Downarrow x$ and $N \Downarrow \lambda y.\, N'$ for some x, y, and N' such that $N' \Downarrow C_{\mathrm{e}}[xV]$, with $y\ \mathcal{R}\ V$ and $z\ \mathcal{R}\ C_{\mathrm{e}}[z]$ for some value V, evaluation context C_{e}, and fresh z.

6. the converse of (5), i.e., $N \Downarrow x$ and $M \Downarrow \lambda y.\, M'$ for some x, y, and M' such that $M' \Downarrow C_{\mathrm{e}}[xV]$, with $V\ \mathcal{R}\ y$ and $C_{\mathrm{e}}[z]\ \mathcal{R}\ z$ for some value V, evaluation context C_{e}, and fresh z.

Then *η-eager normal-form bisimilarity*, $\leftrightarrows_\eta$, is the largest η-eager normal-form bisimulation.

We sometimes call relation $\leftrightarrows_\eta$ the *η-Eager-Tree equality.*

Remark 2.2.7. Definition 2.2.6 coinductively allows η-expansions to occur underneath other η-expansions, hence trees with infinite η-expansions may be equated with finite trees. For instance,

$$x \leftrightarrows_\eta \lambda y.\, xy \leftrightarrows_\eta \lambda y.\, x(\lambda z.\, yz) \leftrightarrows_\eta \lambda y.\, x(\lambda z.\, y(\lambda w.\, zw)) \leftrightarrows_\eta \dots$$

An example of $\leftrightarrows_\eta$ equating a finite tree with an infinite tree is as follows: take a fixpoint combinator Y, and define $f \stackrel{\text{def}}{=} (\lambda zxy.\, x(zy))$. We then have $Yfx \Longrightarrow \lambda y.\, x(Yfy)$, and then $x(Yfy) \Longrightarrow x(\lambda z.\, y(Yfz))$, and so on. Hence, we have $x \leftrightarrows_\eta Yfx$.

2.2.2 The original encodings

Milner noticed [Mil90a, Mil92] that his call-by-value encoding can be easily tuned so to mimic forms of evaluation in which, in an application MN, the function M is run first, or the argument N is run first, or function and argument are run in parallel (the proofs are actually carried out for this last option). We chose here the first scenario, because it is more in line with ordinary call-by-value. A discussion on the 'parallel' call-by-value is deferred to Section 3.2.1.

The core of any encoding of the λ-calculus into a process calculus is the translation of function application. This becomes a particular form of parallel combination of two processes, the function and its argument; β_{v}-reduction is then modeled as process interaction.

The encoding of a λ-term is parametric over a name; this may be thought of as the *location* of that term, or as its *continuation*. A term that becomes a value signals so at its continuation name and, in doing so, it grants access to the body of the value. Such body is replicated, so that the value may be copied several times. When the value is a function, its body can receive two names: (the access to) its value-argument, and the following continuation. In the translation of application, first the function is run, then the argument; finally the function is informed of its argument and continuation.

In the original paper [Mil90a], Milner presented two candidates for the encoding of call-by-value λ-calculus [Plo75]. They follow the same idea, but with a technical difference in the rule for variables. One encoding, $\mathcal{V}$, is defined as follows (for the case of application,

we adapt the encoding from parallel call-by-value to left-to-right call by value, as described above):

$$\begin{aligned}
\mathcal{V}[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} (p)\ \overline{p}(y).\, !y(x,q).\, \mathcal{V}[\![M]\!]\langle q\rangle \\
\mathcal{V}[\![MN]\!] &\stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q\,)(\mathcal{V}[\![M]\!]\langle q\rangle \mid q(y).\, \boldsymbol{\nu} r\ (\mathcal{V}[\![N]\!]\langle r\rangle \mid r(w).\, \overline{y}\langle w,p\rangle)) \\
\mathcal{V}[\![x]\!] &\stackrel{\text{def}}{=} (p)\ \overline{p}\langle x\rangle
\end{aligned}$$

In the other encoding, $\mathcal{V}'$, application and λ-abstraction are treated as in $\mathcal{V}$; the rule for variables is:

$$\mathcal{V}'[\![x]\!] \stackrel{\text{def}}{=} (p)\ \overline{p}(y).\, !y(z,q).\, \overline{x}\langle z,q\rangle\ .$$

The encoding $\mathcal{V}$ is more efficient than $\mathcal{V}'$. In $\mathcal{V}'$, the encoding of a λ-calculus variable gives rise to a one-place buffer. As the computation proceeds, buffers are chained together, gradually increasing the number of steps necessary to simulate a β-reduction. This phenomenon does not occur in $\mathcal{V}$, where the occurrence of a variable disappears after it is used.

2.2.3 Difficulties with the encodings

The immediate free output in the encoding of variables in $\mathcal{V}$ breaks the validity of β_{v}-reduction; i.e., there exist a term M and a value V such that $\mathcal{V}[\![(\lambda x.\, M)V]\!] \not\approx \mathcal{V}[\![M\{^{V}\!/_{x}\}]\!]$ [San93a]. The encoding $\mathcal{V}'$ fixes this by communicating, instead of a free name, a fresh pointer to that name. Technically, the initial free output of x is replaced by a bound output coupled with a link to x (the process $!y(z,q).\, \overline{x}\langle z,q\rangle$, receiving at y and re-emitting at x). Thus β_{v}-reduction is valid, i.e., $\mathcal{V}[\![(\lambda x.\, M)V]\!] \approx \mathcal{V}[\![M\{^{V}\!/_{x}\}]\!]$ for any M and V [San93a].

(The final version of Milner's paper [Mil92], which appeared in the *Journal of Mathematical Structures in Computer Science*, was written after the results in [San93a] were known, and presents only the encoding $\mathcal{V}'$.)

Nevertheless, $\mathcal{V}'$ only delays the free output, as the added link contains itself a free output. As a consequence, we can show that other desirable equalities of call-by-value are broken in $\mathcal{V}'$. An example is law (5) from the Introduction, as stated by Proposition 2.2.8 below. This law is desirable (and indeed valid for contextual equivalence, or the Eager-Tree equality) intuitively because, in any substitution closure of the law, either both terms diverge, or they converge to the same value (depending on whether the computation resulting from the instantiation of xv diverges or not). The same argument holds for the λ-closures of these terms, $\lambda x.\, xV$ and $\lambda x.\, I(xV)$.

We recall that $\simeq^{\pi}$ is barbed congruence in the π-calculus.

Proposition 2.2.8 (Non-law)**.** *For any value V, we have:*

$$\mathcal{V}'[\![I(xV)]\!] \not\simeq^{\pi} \mathcal{V}'[\![xV]\!] \quad \textit{and} \quad \mathcal{V}[\![I(xV)]\!] \not\simeq^{\pi} \mathcal{V}[\![xV]\!]\ .$$

(The law is violated also under coarser equivalences, such as contextual equivalence.)

Proof. For simplicity, we give the proof when $V = y$, and for encoding $\mathcal{V}$. The same can be shown for an arbitrary value V and for the encoding $\mathcal{V}'$, through similar calculations. We use algebraic laws of the equivalence $\simeq^\pi$, or of its associated proof techniques, to carry out the calculations (cf. [SW01]).

$$\begin{aligned}
\mathcal{V}[\![xy]\!]\langle p\rangle &= \boldsymbol{\nu} q\,(\overline{q}\langle x\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\overline{r}\langle y\rangle \mid r(w).\,\overline{u}\langle w,p\rangle)) \\
&\simeq^\pi \boldsymbol{\nu} r\,(\overline{r}\langle y\rangle \mid r(w).\,\overline{x}\langle w,p\rangle) \\
&\simeq^\pi \overline{x}\langle y,p\rangle \\
\mathcal{V}[\![I(xy)]\!]\langle p\rangle &\simeq^\pi \boldsymbol{\nu} q\,(\mathcal{V}[\![I]\!]\langle q\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{V}[\![xy]\!]\langle r\rangle \mid r(w).\,\overline{u}\langle w,p\rangle)) \\
&\simeq^\pi \boldsymbol{\nu} q\,(\overline{q}(u).\,!u(z,q').\,\overline{q'}\langle z\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{V}[\![xy]\!]\langle r\rangle \mid r(w).\,\overline{u}\langle w,p\rangle)) \\
&\simeq^\pi \boldsymbol{\nu} u\,(!u(z,q).\,\overline{q}\langle z\rangle \mid \boldsymbol{\nu} r\,(\overline{x}\langle y,r\rangle \mid r(w).\,\overline{u}\langle w,p\rangle)) \\
&\simeq^\pi \boldsymbol{\nu} r\,(\overline{x}\langle y,r\rangle \mid r(w).\,\boldsymbol{\nu} u\,(!u(z,q).\,\overline{q}\langle z\rangle \mid \overline{u}\langle w,p\rangle)) \\
&\simeq^\pi \boldsymbol{\nu} r\,(\overline{x}\langle y,r\rangle \mid r(w).\,\overline{p}\langle w\rangle) \\
&= \boldsymbol{\nu} q\,(\overline{x}\langle y,q\rangle \mid q(z).\,\overline{p}\langle z\rangle) \\
&= \boldsymbol{\nu} q\,(\overline{x}\langle y,q\rangle.\,q \triangleright p)
\end{aligned}$$

□

In presence of the normal form xy, the application of the identity I becomes observable. Indeed, in the second term, a fresh name, q, is sent instead of continuation p, and a link between q and p is installed. This corresponds to a law which is valid in ALπ, but not in π.

This problem can be avoided by iterating the transformation that takes us from $\mathcal{V}$ to $\mathcal{V}'$ (i.e., the replacement of a free output with a bound output so to avoid all emissions of free names). Thus the target language becomes Internal π; the resulting encoding is analysed in Section 2.2.6.

Another solution is to control the use of name capabilities in processes. In this case the target language becomes ALπ, and we need not modify the initial encoding $\mathcal{V}$. This situation is analysed in Section 2.2.8.

We moreover notice that in both solutions, the use of link processes validates the following law — a form of η-expansion — (the law fails for Milner's encoding into the π-calculus):

$$\lambda y.\,xy = x$$

In the call-by-value λ-calculus this is a useful sensible law (that holds because substitutions replace variables with values).

Remark 2.2.9 (Generalisations of law (5))**.** As a direct consequence of Proposition 2.2.8, for any evaluation context C_{e} and any value V, we have

$$\mathcal{V}[\![C_{\mathrm{e}}[I(xV)]]\!]\langle p\rangle \not\simeq^\pi \mathcal{V}[\![C_{\mathrm{e}}[xV]]\!]\langle p\rangle \ .$$

$$
\begin{aligned}
\mathcal{I}[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} (p)\ \overline{p}(y).\, !y(x,q).\, \mathcal{I}[\![M]\!]\langle q\rangle \\
\mathcal{I}[\![x]\!] &\stackrel{\text{def}}{=} (p)\ \overline{p}(y).\, y \triangleright x \\
\mathcal{I}[\![MN]\!] &\stackrel{\text{def}}{=} (p)\ \boldsymbol{\nu} q\, \big(\mathcal{I}[\![M]\!]\langle q\rangle \mid q(y).\, \boldsymbol{\nu} r\, \big(\mathcal{I}[\![N]\!]\langle r\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p)\big)\big)
\end{aligned}
$$

Figure 2.2: The encoding into Iπ

One may want to generalize further this law, by replacing the identity I with an aribtrary term $\lambda z.\, M$, provided M somehow "uses" z: we may take, for instance, $C'_{\mathrm{e}}[z]$ for some evaluation context C'_{e}. We can then show, indeed:

$$\mathcal{V}[\![C_{\mathrm{e}}[(\lambda z.\, C'_{\mathrm{e}}[z])(xV)]]\!]\langle p\rangle \not\approx^{\pi} \mathcal{V}[\![C_{\mathrm{e}}[C'_{\mathrm{e}}[xV]]]\!]\langle p\rangle \ .$$

This would still hold by replacing $C'_{\mathrm{e}}[z]$ with any term that reduces to $C'_{\mathrm{e}}[z]$. Generalizing further is outside the scope of this thesis. The work by Accattoli and Sacerdoti Coen [AC15] on "useful reductions" could provide hints to investigate this.

2.2.4 Definition of the encoding

The encoding uses two kinds of names: *triggers* $x, y, v, w \ldots$ and *continuations* $p, q, r, \ldots$. For simplicity, we assume that the set of trigger names contains the set of λ-variables. Continuation names are always used linearly; more precisely, they are only used once, in subject position. On the other hand, trigger names may be used multiple times, and may occur under a replication. This is a very mild form of typing. We could avoid the distinction between these two kinds of names, at the cost of introducing additional replications in the encoding.

Figure 2.2 presents the encoding into Iπ, which is derived from Milner's encoding by removing the free outputs as explained in Section 2.2.3. Process $a \triangleright b$ represents a *link*, sometimes called forwarder; for readability we have adopted the infix notation $a \triangleright b$ for the constant $\triangleright$. It transforms all outputs at a into outputs at b (therefore a, b are names of the same sort). Thus the body of $a \triangleright b$ is replicated, unless a and b are *continuation names* (names such as p, q, r over which the encoding of a term is abstracted). The definition of the constant $\triangleright$ therefore is:

$$
\triangleright \ \stackrel{\triangle}{=} \ \begin{cases} (p,q)\ p(x).\, \overline{q}(y).\, y \triangleright x \\ \qquad\qquad \text{if } p, q \text{ are continuation names} \\ (x,y)\ !x(z,p).\, \overline{y}(w,q).\, (q \triangleright p \mid w \triangleright z) \\ \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}
$$

(The distinction between continuation names and the other sorts of names is not necessary, but simplifies the proofs.)

We recall some useful properties of links [San96b]. The first one is a form of composition, expressed in terms of the expansion preorder (Definition 0.3.11).

Lemma 2.2.10. *We have:*

1. $\boldsymbol{\nu} q\,(p \triangleright q \mid q \triangleright r) \succeq p \triangleright r$, *for all continuation names* p, r.
2. $\boldsymbol{\nu} y\,(x \triangleright y \mid y \triangleright z) \succeq x \triangleright z$, *for all trigger names* x, z.

The use of links in the encoding ensures that law (5) (from the Introduction) indeed holds. More generally, the encoding equates $C_{\mathrm{e}}[xV]$ and $(\lambda z.\, C_{\mathrm{e}}[z])(xV)$, for any evaluation context C_{e}, value V and variable x (z is assumed to be fresh in C_{e}):

$$\mathcal{I}[\![(\lambda z.\, C_{\mathrm{e}}[z])(xV)]\!] \approx \mathcal{I}[\![C_{\mathrm{e}}[xV]]\!] \ . \tag{2.1}$$

Law (5) is then an instance of law (2.1), by taking context C_{e} to be simply the hole. Law (2.1) can be established by induction on the evaluation context C_{e}, using algebraic reasoning. We give a simpler proof in Lemma 2.2.22 below (Section 2.2.7.1).

2.2.5 Validity of β_{v}-reduction

The lemmas in this section are formulated using the expansion preorder (Definition 0.3.11), as it is useful for the soundness proof. For the completeness proof, we use these lemmas with $\approx$ in place of $\succeq$.

We first introduce a useful notation:

Definition 2.2.11. Given a value V, we define $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle$ as follows:

1. If $V = x$, then $\mathcal{I}_{\mathrm{V}}[\![x]\!]\langle z\rangle \stackrel{\triangle}{=} z \triangleright x$.
2. If $V = \lambda x.\, M$, then $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \stackrel{\triangle}{=} !z(x, q).\, \mathcal{I}[\![M]\!]\langle q\rangle$.

We observe that for any value V, we have:

$$\mathcal{I}[\![V]\!]\langle p\rangle = \overline{p}(z).\, \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \ .$$

The next lemma shows that, on the processes obtained by the encoding into Iπ, links behave as substitutions. Some technical details of the proofs are found in Appendix D.1.1. We recall that p, q are continuation names, whereas x, y are variable names.

Lemma 2.2.12. *We have:*

1. $\boldsymbol{\nu} x\,(\mathcal{I}[\![M]\!]\langle p\rangle \mid x \triangleright y) \succeq \mathcal{I}[\![M\{y/x\}]\!]\langle p\rangle$;
2. $\boldsymbol{\nu} p\,(\mathcal{I}[\![M]\!]\langle p\rangle \mid p \triangleright q) \succeq \mathcal{I}[\![M]\!]\langle q\rangle$;
3. $\boldsymbol{\nu} y\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid x \triangleright y) \succeq \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle$.

Proof. Laws 1 and 2 are proved by induction on M, using algebraic reasoning and Lemma 2.2.10. Law 1 is needed to show law 2.

3. Law 3 can be derived from laws 1 and 2, by case analysis on V:

   - If $V = \lambda x.\, M$, then $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle = !y(w,p).\, \mathcal{I}[\![M\{w/x\}]\!]\langle p\rangle$; we then have

$$\begin{aligned}\boldsymbol{\nu} y\, (\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid x \triangleright y) &\sim !x(z,q).\, \boldsymbol{\nu} w,p\, (\mathcal{I}[\![M\{w/x\}]\!]\langle p\rangle \mid p \triangleright q \mid w \triangleright z)\\ &\succeq !x(z,q).\, \mathcal{I}[\![M\{z/x\}]\!]\langle q\rangle \text{ (by laws 1 and 2)}\\ &= \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle\end{aligned}$$

   - If $V = z$, then $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle = !y(w,q).\, \overline{z}(w',q').\, (w' \triangleright w \mid q' \triangleright q) = y \triangleright z$; we can then conclude by Lemma 2.2.10.

□

Lemma 2.2.13 (Validity of β_{v}-reduction)**.** *For any M, N in Λ, $M \longrightarrow N$ implies $\mathcal{I}[\![M]\!] \succeq \mathcal{I}[\![N]\!]$.*

Proof. One shows $\mathcal{I}[\![(\lambda x.\, M)\, V]\!] \succeq \mathcal{I}[\![M\{V/x\}]\!]$ exploiting algebraic properties of replication. The result then follows by the compositionality of the encoding and the congruence of $\succeq$. □

2.2.6 Soundness of the encoding

The structure of the proof of soundness of the encoding is similar to that for the analogous property for Milner's call-by-name encoding with respect to Levy-Longo Trees [San00]. The details are however different, as in call-by-value both the encoding and the trees (the Eager Trees extended to handle η-expansion) are more complex. As previously, some technical details of the proofs are found in Appendix D.1.2.

Optimised encoding. In order to establish operational correspondence for the encoding, we rely on the expansion preorder, $\succeq$. To be able to reason up to $\succeq$, we need, whenever $M \Longrightarrow N$, the encoding of N to be 'faster' than the encoding of M: the encoding of N should perform less internal computation steps before a visible transition. This property is not satisfied by encoding $\mathcal{I}$.

We therefore introduce an optimised encoding, written $\mathcal{O}$, between λ-terms and Iπ-terms. The encoding is presented in Figure 2.3. In the figure we assume that rules for $\mathcal{O}[\![V]\!]$ and $\mathcal{O}[\![(\lambda x.\, M)V]\!]$ have priority over the others; in other words, in the other rules terms M and N should not be values.

The optimised encoding of Figure 2.3 is obtained from that in Figure 2.2 by performing a few (deterministic) reductions, at the price of a more complex definition. Precisely, in the encoding of application we remove some of the initial communications, including those with which a term signals to have become a value. To achieve this, the encoding of an application goes by a case analysis (4 cases) on the occurrences of values in the subterms.

The general idea of the optimized encoding can be illustrated on two of its defining clauses. For $\mathcal{O}[\![VM]\!]$, the corresponding equation is the result of unfolding the original encoding, and performing one (deterministic) communication. In the case of $\mathcal{O}[\![xV]\!]$, not

$$
\begin{array}{rcl}
\mathcal{O}[\![xV]\!] & \stackrel{\text{def}}{=} & (p)\ \overline{x}(z,q).\,(\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q \triangleright p) \\
\mathcal{O}[\![(\lambda x.\,M)V]\!] & \stackrel{\text{def}}{=} & (p)\ \boldsymbol{\nu} y, w\,(\mathcal{O}_{\mathrm{V}}[\![\lambda x.\,M]\!]\langle y\rangle \mid \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(w', r').\,(w' \triangleright w \mid r' \triangleright p)) \\
\mathcal{O}[\![VM]\!] & \stackrel{\text{def}}{=} & (p)\ \boldsymbol{\nu} y\,(\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid \boldsymbol{\nu} r\,(\mathcal{O}[\![M]\!]\langle r\rangle \mid r(w).\,\overline{y}(w', r').\,(w' \triangleright w \mid r' \triangleright p))) \\
\mathcal{O}[\![MV]\!] & \stackrel{\text{def}}{=} & (p)\ \boldsymbol{\nu} q\,(\mathcal{O}[\![M]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} w\,(\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(w', r').\,(w' \triangleright w \mid r' \triangleright p))) \\
\mathcal{O}[\![MN]\!] & \stackrel{\text{def}}{=} & (p)\ \boldsymbol{\nu} q\,(\mathcal{O}[\![M]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} r\,(\mathcal{O}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{y}(w', r').\,(w' \triangleright w \mid r' \triangleright p))) \\
\mathcal{O}[\![V]\!] & \stackrel{\text{def}}{=} & (p)\ \overline{p}(y).\,\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle y\rangle
\end{array}
$$

where $\mathcal{O}_{\mathrm{V}}[\![V]\!]$ is thus defined :

$$
\begin{array}{rcl}
\mathcal{O}_{\mathrm{V}}[\![\lambda x.\,M]\!] & \stackrel{\text{def}}{=} & (y)\ !y(x,q).\,\mathcal{O}[\![M]\!]\langle q\rangle \\
\mathcal{O}_{\mathrm{V}}[\![x]\!] & \stackrel{\text{def}}{=} & (y)\ y \triangleright x
\end{array}
$$

In the rules, we suppose that M and N are not values.

Figure 2.3: Optimised encoding into Iπ

only do we unfold the original encoding and reduce along deterministic communications, but we also perform the administrative reductions that always precede the execution of $\mathcal{I}[\![xV]\!]$, as shown in the following calculations

$$
\begin{array}{rcl}
\mathcal{I}[\![xV]\!]\langle p\rangle & = & \boldsymbol{\nu} q\,(\overline{q}(y).\,\mathcal{I}_{\mathrm{V}}[\![x]\!]\langle y\rangle \mid q(y).\,\boldsymbol{\nu} r\,(\overline{r}(w).\,\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid r(w).\,\overline{y}(w', p').\,(w' \triangleright w \mid p' \triangleright p))) \\
& \succeq & \boldsymbol{\nu}(y,w)\,(\mathcal{I}_{\mathrm{V}}[\![x]\!]\langle y\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(w', p').\,(w' \triangleright w \mid p' \triangleright p))\ \text{(algebraic laws)} \\
& = & \boldsymbol{\nu}(y,w)\,(y \triangleright x \mid \overline{y}(w', p').\,(w' \triangleright w \mid p' \triangleright p) \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle) \\
& \succeq & \boldsymbol{\nu}(w, w', p')\,(\overline{x}(z,q).\,(z \triangleright w' \mid q \triangleright p') \mid p' \triangleright p \mid w' \triangleright w \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle) \\
& \succeq & \overline{x}(z,q).\,\boldsymbol{\nu}(w, w', p')\,.\,(z \triangleright w' \mid w' \triangleright w \mid q \triangleright p' \mid p' \triangleright p \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle) \\
& \succeq & \overline{x}(z,q).\,\boldsymbol{\nu} w\,(q \triangleright p \mid z \triangleright w \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle)\ \ \text{(by Lemma 2.2.10)} \\
& \succeq & \overline{x}(z,q).\,(q \triangleright p \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle)\ \ \text{(by Lemma 2.2.12)}
\end{array}
$$

The definition of $\mathcal{O}_{\mathrm{V}}[\![xV]\!]$ is obtained by replacing $\mathcal{I}_{\mathrm{V}}[\![V]\!]$ with $\mathcal{O}_{\mathrm{V}}[\![V]\!]$.

Therefore, in the optimised encoding, the term in evaluation position is always ready to be fired. As a consequence, in the encoding of $C_{\mathrm{e}}[xV]$, the first transition is the output on x due to xV, rather than administrative reductions that may occur in C_{e}. Similarly, the τ-transition from $C_{\mathrm{e}}[(\lambda x.\,M)V]$ corresponds to the β_{v}-reduction occuring in evaluation position.

Relating the encodings. Lemma 2.2.15 below uses expansion to state that encoding $\mathcal{O}$ is indeed an optimised version of $\mathcal{I}$. It is established by analysing the behaviour of the encoding of a stuck redex according to $\mathcal{O}$:

Lemma 2.2.14. *We have:*

$$\mathcal{O}[\![C_{\mathrm{e}}[xV]]\!]\langle p\rangle \succeq \overline{x}(z,q).\,(\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle).$$

Proof. By induction on the evaluation context C_{e}. The base case is treated using the algebraic calculations which are given above. □

Lemma 2.2.15. *For all $M \in \Lambda$, $\mathcal{I}[\![M]\!] \succeq \mathcal{O}[\![M]\!]$.*

This lemma is proved using simple algebraic laws.

Operational correspondence. We formulate operational correspondence between λ-terms and their encodings using the optimised encoding.

In the lemma below, recall that we identify processes or transitions that only differ in the choice of the bound names. Therefore, when we say a process has exactly one immediate transition, we mean that there is a unique pair (μ,P), up to alpha-conversion of the pair, such that $\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\mu} P$.

Lemma 2.2.16 (Operational correspondence)**.** *For any $M \in \Lambda$ and fresh p, process $\mathcal{O}[\![M]\!]\langle p\rangle$ has exactly one immediate transition, and exactly one of the following clauses holds:*

1. *$\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\overline{p}(y)} P$ and M is a value, with $P = \mathcal{O}_{\mathrm{V}}[\![M]\!]\langle y\rangle$;*
2. *$\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\overline{x}(z,q)} P$ and $M = C_{\mathrm{e}}[xV]$ and*
$$P \succeq \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle;$$
3. *$\mathcal{O}_{\mathrm{V}}[\![M]\!]\langle p\rangle \xrightarrow{\tau} P$ and there is N with $M \longrightarrow N$ and $P \succeq \mathcal{O}[\![N]\!]\langle p\rangle$.*

Proof. We reason by induction on M, and rely on Lemmas 2.2.13, 2.2.10, and 2.2.12. □

The operational correspondence has two immediate consequences, for converging and diverging terms.

Corollary 2.2.17. *If $\mathcal{O}[\![M]\!]\langle p\rangle \Rightarrow\!\xrightarrow{\mu} P$ and $\mu \neq \tau$, then M admits an eager normal-form M' such that $\mathcal{O}[\![M']\!]\langle p\rangle \xrightarrow{\mu}\!\succeq P$.*

Proof. By induction on the length of the reduction $\mathcal{O}[\![M]\!]\langle p\rangle \Rightarrow\!\xrightarrow{\mu} P$. If $\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\mu} P$ and $\mu \neq \tau$, by Lemma 2.2.16, M is an eager normal-form. Otherwise, there is P' such that $\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\tau} P' \stackrel{\mu}{\Rightarrow} P$; by Lemma 2.2.16, there is N such that $M \longrightarrow N$, and $P' \succeq \mathcal{O}[\![N]\!]\langle p\rangle$. Therefore $\mathcal{O}[\![N]\!]\langle p\rangle \stackrel{\mu}{\Rightarrow} Q$, where $Q \succeq P$. We can then apply the induction hypothesis to deduce that N admits an eager normal-form M', which is also an eager normal-form of M, with $\mathcal{O}[\![M']\!]\langle p\rangle \xrightarrow{\mu}\!\succeq Q \succeq P$. □

Lemma 2.2.18. *$\mathcal{O}[\![M]\!]\langle p\rangle \approx \mathbf{0}$ iff $M \Uparrow$.*

Proof. Suppose $\mathcal{O}[\![M]\!] \not\approx (p)\ \mathbf{0}$. Then there is P, p and $\mu \neq \tau$ such that $\mathcal{O}[\![M]\!]\langle p\rangle \stackrel{\mu}{\Rightarrow} \mathcal{O}[\![P]\!]\langle p\rangle$, and by Corollary 2.2.17, M has an eager normal form (hence M does not diverge).

For the converse implication, assume $\mathcal{O}[\![M]\!] \approx (p)\ \mathbf{0}$. By Corollary 2.2.16, $\mathcal{O}[\![M]\!]\langle p\rangle \longrightarrow P$ for some P, and there is N such that $M \longrightarrow N$, and $\mathcal{O}[\![N]\!]\langle p\rangle \approx P \approx (p)\ \mathbf{0}$. With this property, using coinduction we derive $M \Uparrow$. □

Soundness. Operational correspondence allows us to show that the observables for bisimilarity in the encoding π-terms imply the observables for η-eager normal-form bisimilarity in the encoded λ-terms. The delicate cases are those in which a branch in the tree of the terms is produced — case (2) of Definition 2.2.3 — and where an η-expansion occurs — thus a variable is equivalent to an abstraction, cases (5) and (6) of Definition 2.2.6.

For the branching, we exploit a decomposition property on π-terms, roughly allowing us to derive from the bisimilarity of two parallel compositions the componentwise bisimilarity of the single components. For the η-expansion, if $\mathcal{I}[\![x]\!] \approx \mathcal{I}[\![\lambda z.\, M]\!]$, where $M \Downarrow C_{\mathrm{e}}[xV]$, we use a coinductive argument to derive $V \leftrightarrows_\eta z$ and $C_{\mathrm{e}}[y] \leftrightarrows_\eta y$, for y fresh; from this we then obtain $\lambda z.\, M \leftrightarrows_\eta x$.

The following lemma corresponds to the case of the branching; it allows us to decompose an equivalence between two parallel processes. This result is used to handle equalities of the form $\mathcal{I}[\![C_{\mathrm{e}}[xV]]\!] \approx \mathcal{I}[\![C'_{\mathrm{e}}[xV']]\!]$, in order to deduce equivalence between V and V' on the one hand, and between $C_{\mathrm{e}}[y]$ and $C'_{\mathrm{e}}[y]$ on the other.

Lemma 2.2.19. *Suppose that a does not occur free in Q or Q', and one of the following holds:*

1. $a(x).\, P \mid Q \approx a(x).\, P' \mid Q'$;
2. $!a(x).\, P \mid Q \approx !a(x).\, P' \mid Q'$.

Then we also have $Q \approx Q'$.

Proof. Since a does not appear free in Q or Q', neither of those processes can perform an action in which a occurs free (here we rely on the labelled transition system being *ground*). As a consequence, we can play a bisimulation game relating Q and Q', which can be deduced from the game between $a(x).\, P \mid Q$ and $a(x).\, P' \mid Q'$ (resp. from the game between $!a(x).\, P \mid Q$ and $!a(x).\, P' \mid Q'$). □

We now show that the only λ-terms whose encoding is bisimilar to $\mathcal{I}[\![x]\!]$ reduce either to x, or to a (possibly infinite) η-expansion of x.

Lemma 2.2.20. *If V is a value and x a variable, $\mathcal{O}_{\mathrm{V}}[\![V]\!] \approx \mathcal{O}_{\mathrm{V}}[\![x]\!]$ implies that either $V = x$ or $V = \lambda z.\, M$, where the eager normal form of M is of the form $C_{\mathrm{e}}[xV']$, with $\mathcal{O}_{\mathrm{V}}[\![V']\!] \approx \mathcal{O}_{\mathrm{V}}[\![z]\!]$ and $\mathcal{O}[\![C_{\mathrm{e}}[y]]\!] \approx \mathcal{O}[\![y]\!]$ for any y fresh.*

Proof. By definition, $\mathcal{O}[\![x]\!]\langle p\rangle = \overline{p}(y).\, y \triangleright x$. If $y \neq x$, then $\mathcal{I}[\![y]\!] \not\approx \mathcal{I}[\![x]\!]$; therefore if $\mathcal{I}[\![V]\!] \approx \mathcal{I}[\![x]\!]$ and $V \neq x$, then V is necessarily an abstraction, say $V = \lambda z.\, M$. The following equalities then hold:

- $\mathcal{O}[\![\lambda z.\, M]\!]\langle p\rangle = \overline{p}(y).\, !y(z,q).\, \mathcal{O}[\![M]\!]\langle q\rangle$, and
- $\mathcal{O}[\![x]\!]\langle p\rangle = \overline{p}(y).\, !y(z,q).\, \overline{x}(z',q').\, (z' \triangleright z \mid q' \triangleright q)$.

Therefore $\mathcal{O}[\![M]\!]\langle q\rangle \approx \overline{x}(z', q').\,(z' \triangleright z \mid q' \triangleright q)$, and by Corollary 2.2.17 and Lemma 2.2.16, M has an eager normal form $C_{\mathrm{e}}[xV']$. We have, using Lemma 2.2.14:

$$\begin{aligned}\mathcal{O}[\![M]\!]\langle q\rangle &\approx \mathcal{O}[\![C_{\mathrm{e}}[xV']]\!]\langle q\rangle \\ &\approx \overline{x}(z', q').\,(\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z'\rangle \mid q'(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle q\rangle)\ ,\end{aligned}$$

hence

$$\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z'\rangle \mid q'(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle q\rangle \approx z' \triangleright z \mid q' \triangleright q.$$

Then, z' is not free in $q'(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle q\rangle$ and q' is not free in $\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z'\rangle$. Furthermore, $\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z'\rangle$ is prefixed by an input on z'. By applying Lemma 2.2.19 twice, we deduce

$$\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z'\rangle \approx z' \triangleright z \quad \text{and} \quad q'(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle q\rangle \approx q' \triangleright q\ .$$

By definition, $z' \triangleright z = \mathcal{O}_{\mathrm{V}}[\![z]\!]\langle z'\rangle$, so

$$\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z'\rangle \approx \mathcal{O}_{\mathrm{V}}[\![z]\!]\langle z'\rangle\ .$$

Moreover, by definition of $\triangleright$, we have

$$q'(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle q\rangle \approx q'(y).\,\overline{q}(y').\,(y' \triangleright y)\ .$$

The latter equivalence gives, by playing a step in the bisimulation game

$$\begin{aligned}\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle q\rangle &\approx \overline{q}(y').\,(y' \triangleright y) \\ &= \mathcal{O}[\![y]\!]\langle q\rangle\ .\end{aligned}$$

□

We can now establish soundness of the encoding.

Proposition 2.2.21 (Soundness)**.** *For any $M, N \in \Lambda$, if $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$ then $M \leftrightarrows_\eta N$.*

Proof. Let $\mathcal{R} \stackrel{\text{def}}{=} \{(M, N) \mid \mathcal{O}[\![M]\!] \approx \mathcal{O}[\![N]\!]\}$. We show that $\mathcal{R}$ is an η-eager normal-form bisimulation, and conclude by Lemma 2.2.15. Assume $\mathcal{O}[\![M]\!] \approx \mathcal{O}[\![N]\!]$.

1. Suppose w.l.o.g. that $M \Uparrow$. Then by Lemma 2.2.18, this gives $\mathcal{O}[\![M]\!]\langle p\rangle \approx \mathbf{0}$. Hence by hypothesis $\mathcal{O}[\![N]\!]\langle p\rangle \approx \mathbf{0}$, which gives, by Lemma 2.2.18 that $N \Uparrow$.

2. Otherwise, M and N have eager normal-forms M' and N'; thus $M \Downarrow M'$ and $N \Downarrow N'$. Therefore by Lemma 2.2.15 and validity of β_{v}-reduction, $\mathcal{O}[\![M']\!] \approx \mathcal{O}[\![N']\!]$. Since M' is in eager normal-form, by Lemma 2.2.16, either $\mathcal{O}[\![M']\!]\langle p\rangle \xrightarrow{\overline{x}(z,q)} P$ or $\mathcal{O}[\![M']\!]\langle p\rangle \xrightarrow{\overline{p}(y)} P$, and likewise for N'. This gives rise to two cases:

(a) $M' = C_{\mathrm{e}}[xV]$ and $N' = C'_{\mathrm{e}}[xV']$, and

$$\begin{aligned}&\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle\\ &\approx \mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z\rangle \mid q(y).\,\mathcal{O}[\![C'_{\mathrm{e}}[y]]\!]\langle p\rangle\ .\end{aligned}$$

Name q does not appear free in $\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle$ or $\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z\rangle$, hence by Lemma 2.2.19

$$\mathcal{O}_{\mathrm{V}}[\![V]\!] \approx \mathcal{O}_{\mathrm{V}}[\![V']\!]\ .$$

Likewise, z' does appear free neither in $q(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle$ nor in $q(y).\,\mathcal{O}[\![C'_{\mathrm{e}}[y]]\!]\langle p\rangle$, and both $\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle$ and $\mathcal{O}_{\mathrm{V}}[\![V']\!]\langle z\rangle$ are prefixed by a replicated input on z. Hence, by Lemma 2.2.19

$$\mathcal{O}[\![C_{\mathrm{e}}[y]]\!] \approx \mathcal{O}[\![C'_{\mathrm{e}}[y]]\!]\ .$$

Therefore $V\ \mathcal{R}\ V'$ and $C_{\mathrm{e}}[y]\ \mathcal{R}\ C'_{\mathrm{e}}[y]$.

(b) M' and N' are values. They can be abstractions or variables.

i. If both are abstractions $M' = \lambda z.\,M''$, $N' = \lambda z.\,N''$, and $!y(z,q).\,\mathcal{O}[\![M'']\!]\langle q\rangle \approx !y(z,q).\,\mathcal{O}[\![N'']\!]\langle q\rangle$, hence $\mathcal{O}[\![M'']\!] \approx \mathcal{O}[\![N'']\!]$, and $M''\ \mathcal{R}\ N''$.

ii. If both are variables, we must have $M' = N' = x$: for any $y \neq x$, $\mathcal{O}[\![x]\!] \not\approx \mathcal{O}[\![y]\!]$. We have $x\ \mathcal{R}\ x$ thus we can conclude.

iii. Otherwise, assume $M' = \lambda z.\,M''$ and $N' = x$ without loss of generality. Then $\mathcal{O}[\![M']\!] \approx \mathcal{O}[\![N']\!] \approx \mathcal{O}[\![x]\!]$. By Lemma 2.2.20, $M'' \Downarrow C_{\mathrm{e}}[xV]$ for some C_{e}, V, and also $\mathcal{O}[\![V]\!] \approx \mathcal{O}[\![z]\!]$ and $\mathcal{O}[\![C_{\mathrm{e}}[y]]\!] \approx \mathcal{O}[\![y]\!]$ for any y fresh. Hence, $V\ \mathcal{R}\ z$, $C_{\mathrm{e}}[y]\ \mathcal{R}\ y$ for some y fresh, and we can conclude by case 6 of Definition 2.2.6.

□

2.2.7 Completeness of the encoding

Suppose $M \leftrightarrows_\eta N$. Then there is an η-eager normal-form bisimulation $\mathcal{R}$ such that $M\mathcal{R}N$. The completeness of the encoding can thus be stated as follows: given $\mathcal{R}$ an η-eager normal-form bisimulation, for all $(M,N) \in \mathcal{R}$, $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$.

To ease the reader into the proof, we first show the completeness for $\leftrightarrows$, rather than $\leftrightarrows_\eta$. The presentation of the extension to $\leftrightarrows_\eta$ is deferred to Section 2.2.7.2.

2.2.7.1 Completeness for $\leftrightarrows$

The system of equations. Suppose $\mathcal{R}$ is an eager normal-form bisimulation. We define an (infinite) system of equations $\mathcal{E}_\mathcal{R}$, solutions of which will be obtained from the encodings of the pairs in $\mathcal{R}$. We then use Theorem 1.4.11 and Lemma 1.1.14 to show that $\mathcal{E}_\mathcal{R}$ has a unique solution.

We assume an ordering on names and variables, so to be able to view (finite) sets of these as tuples. Moreover, if F is an abstraction, say $(\widetilde{a})\ P$, then $(\widetilde{y})\ F$ is an abbreviation for the abstraction $(\widetilde{y},\widetilde{a})\ P$.

In system $\mathcal{E}_{\mathcal{R}}$, there is one equation $X_{M,N} = E_{M,N}$ for each pair $(M,N) \in \mathcal{R}$. The body $E_{M,N}$ essentially describes the encoding of the eager normal form (or absence thereof) of M and N, with the variables of the equations representing the coinductive hypotheses. To formalise this, we extend the encoding of the λ-calculus to equation variables by setting

$$\mathcal{I}[\![X_{M,N}]\!] \stackrel{\text{def}}{=} (p)\ X_{M,N}\langle \widetilde{y}, p\rangle \qquad \text{where } \widetilde{y} = \mathsf{fv}(M,N)\ .$$

We now describe the equation $X_{M,N} = E_{M,N}$, for $(M,N) \in \mathcal{R}$. The equation is parametrised on the free variables of M and N (to ensure that the body $E_{M,N}$ is a name-closed abstraction) together with an additional continuation name (as all encodings of terms). Below, we let $\widetilde{y}$ stand for $\mathsf{fv}(M,N)$.

1. If $M \Downarrow x$ and $N \Downarrow x$, then the equation is the encoding of x:

   $$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![x]\!]$$

   that is, the equation is

   $$X_{M,N} = (\widetilde{y}, p)\ \overline{p}(z).\, z \triangleright x$$

   Since x is the eager normal-form of M and N, $x \in \widetilde{y}$. Note that $\widetilde{y}$ can contain more names, occurring free in M or N.

2. If $M \Uparrow$ and $N \Uparrow$, then the equation uses a purely-divergent term. We choose the encoding of Ω for this:

   $$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\Omega]\!]$$

   Note that since the encoding of any diverging term is bisimilar to $\mathbf{0}$, we could replace the body of this equation with $(\widetilde{y}, p)\ \mathbf{0}$.

3. If $M \Downarrow \lambda x.\, M'$ and $N \Downarrow \lambda x.\, N'$, then the equation encodes an abstraction whose body refers to the normal forms of M', N', via the variable $X_{M',N'}$:

   $$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\lambda x.\, X_{M',N'}]\!]$$

   that is, the equation is

   $$X_{M,N} = (\widetilde{y}, p)\ \overline{p}(z).\, !z(x,q).\, X_{M',N'}\langle \widetilde{y}', q\rangle$$

4. If $M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C'_{\mathrm{e}}[xV']$, we separate the evaluation contexts and the values, as in Definition 2.2.3. In the body of the equation, this is achieved by: (i) rewriting $C_{\mathrm{e}}[xV]$ into $(\lambda z.\, C_{\mathrm{e}}[z])(xV)$, for some fresh z, and similarly for C'_{e} and V' (such a transformation is valid for $\leftrightarrows$, as per law 2.1, which is established in Lemma 2.2.22); and (ii) referring to the equation variable associated to the evaluation contexts, $X_{C_{\mathrm{e}}[z],C'_{\mathrm{e}}[z]}$, and to the equation variable associated to the values, $X_{V,V'}$. This yields the following equation, for z fresh:

   $$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![(\lambda z.\, X_{C_{\mathrm{e}}[z],C'_{\mathrm{e}}[z]})\ (x\ X_{V,V'})]\!]$$

$$
\begin{array}{ll}
M \Uparrow \text{ and } N \Uparrow: & X_{M,N} = (\widetilde{y}, p)\ \mathcal{I}[\![\Omega]\!]\langle p\rangle \\
M \Downarrow x \text{ and } N \Downarrow x: & X_{M,N} = (\widetilde{y}, p)\ \overline{p}(y).\, y \triangleright x \\
M \Downarrow \lambda x.\, M' \text{ and } N \Downarrow \lambda x.\, N': & X_{M,N} = (\widetilde{y}, p)\ \overline{p}(y).\, !y(x,q) X_{M',N'}\langle \widetilde{y}', q\rangle \\
M \Downarrow C_{\mathrm{e}}[xV] \text{ and } N \Downarrow C'_{\mathrm{e}}[xV']: & X_{M,N} = (\widetilde{y}, p)\ \mathcal{I}[\![(\lambda z.\, X_{C_{\mathrm{e}}[z], C'_{\mathrm{e}}[z]})(x X_{V,V'})]\!]\langle p\rangle
\end{array}
$$

Figure 2.4: System $\mathcal{E}_{\mathcal{R}}$ of equations

Figure 2.4 sums up what has been presented, by providing the complete definition of system $\mathcal{E}_{\mathcal{R}}$ (to preserve readability, we do not give the complete expression in the equation for $C_{\mathrm{e}}[xV]$).

As an example to illustrate the construction of $\mathcal{E}_{\mathcal{R}}$, suppose $(I, \lambda x.\, M) \in \mathcal{R}$, where $I = \lambda x.\, x$ and $M = (\lambda zy.\, z)xx'$. The free variables of M are x and x'. We obtain the following equations (assuming x is before x' in the ordering of variables):

$$X_{I,\lambda x.M} = (x', p)\ \overline{p}(y).\, !y(x,q).\, X_{x,M}\langle x, x', q\rangle$$

(coming from $(x')\ \mathcal{I}[\![\lambda x.\, X_{x,M}]\!]$), and

$$X_{x,M} = (x, x', p)\ \overline{p}(y).\, y \triangleright x$$

(coming from $(x, x')\ \mathcal{I}[\![x]\!]$).

Solutions of $\mathcal{E}_{\mathcal{R}}$. Having constructed $\mathcal{E}_{\mathcal{R}}$, the system of equations for $\mathcal{R}$, we now define solutions for $\mathcal{E}_{\mathcal{R}}$ from the encoding of the pairs in $\mathcal{R}$.

We can view the relation $\mathcal{R}$ as an ordered sequence of pairs (e.g., assuming some lexicographical ordering). Then $\mathcal{R}_i$ indicates the tuple obtained by projecting the pairs in $\mathcal{R}$ onto the i-th component ($i = 1, 2$). Moreover we let (M_j, N_j) stand for the j-th pair in $\mathcal{R}$, and $\widetilde{y}_j$ for $\mathsf{fv}(M_j, N_j)$.

$\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ is defined as the sequence of closed abstractions resulting from the encoding of $\mathcal{R}_1$, i.e., the tuple whose j-th component is $(\widetilde{y}_j)\ \mathcal{I}[\![M_j]\!]$, and similarly for $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$.

We observe that if $\widetilde{y} = \mathsf{fv}(M, N)$, then $(\widetilde{y}, p)\ \mathcal{I}[\![M]\!]\langle p\rangle$ and $(\widetilde{y}, p)\ \mathcal{I}[\![N]\!]\langle p\rangle$ are closed abstractions.

To prove that $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of the system, we need to establish law (2.1):

Lemma 2.2.22. *If C_{e} is an evaluation context, V is a value, x is a name and z is fresh in C_{e}, then*

$$\mathcal{I}[\![C_{\mathrm{e}}[xV]]\!] \ \approx\ \mathcal{I}[\![(\lambda z.\, C_{\mathrm{e}}[z])(xV)]\!]\ .$$

Proof. Lemmas 2.2.14 and 2.2.15; give us:

$$\mathcal{I}[\![C_{\mathrm{e}}[xV]]\!]\langle p\rangle \approx \overline{x}(z,q).\, (\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\, \mathcal{I}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle)$$

and

$$\mathcal{I}[\![(\lambda w.\, C_{\mathrm{e}}[w])(xV)]\!]\langle p\rangle \approx \overline{x}(z,q).\,(\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![(\lambda w.\, C_{\mathrm{e}}[w])y]\!]\langle p\rangle)$$

We conclude by validity of β-reduction (Lemma 2.2.13) applied to $\mathcal{I}[\![(\lambda w.\, C_{\mathrm{e}}[w])y]\!]\langle p\rangle$. □

Lemma 2.2.23. *$\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of the system of equations $\mathcal{E}_{\mathcal{R}}$.*

Proof. We only establish the property for $\mathcal{R}_1$, the case for $\mathcal{R}_2$ is handled similarly.

We show that each component of $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ is solution of the corresponding equation, i.e., for the j-th component we show $(\widetilde{y_j})\ \mathcal{I}[\![M_j]\!] \approx E_{M_j,N_j}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]]$.

We reason by cases over the shape of the eager normal form of M_j, N_j.

- If $M_j \Uparrow$, we use Lemma 2.2.18, which gives us $(\widetilde{y_j})\ \mathcal{I}[\![M]\!]_j \approx (\widetilde{y_j}, p)\ \mathbf{0} \approx (\widetilde{y_j})\ \mathcal{I}[\![\Omega]\!]$.
- If $M_j \Downarrow C_{\mathrm{e}}[xV]$, we have to show that:

  $$\mathcal{I}[\![M_j]\!] \approx \mathcal{I}[\![(\lambda z.\, C_{\mathrm{e}}[z])(xV)]\!]\ .$$

  By Lemma 2.2.13, $\mathcal{I}[\![M]\!]_j \approx \mathcal{I}[\![C_{\mathrm{e}}[xV]]\!]$; we then conclude by Lemma 2.2.22.
- If $M_j \Downarrow \lambda x.\, M'$ (and N_j also reduces to an abstraction), then:

  $$\begin{aligned} E_{M_j,N_j}[\mathcal{I}[\![\mathcal{R}_1]\!]]\langle \widetilde{y_j}, p\rangle &= \overline{p}(z).\,!z(x,q).\,\mathcal{I}[\![M']\!]\langle q\rangle \\ &\approx \mathcal{I}[\![M_j]\!]\langle \widetilde{y_j}, p\rangle \text{ (by Lemma 2.2.13)}\ . \end{aligned}$$
- If $M_j \Downarrow x$ (and $N_j \Downarrow x$), again:

  $$\begin{aligned} E_{M_j,N_j}[\mathcal{I}[\![\mathcal{R}_1]\!]]\langle \widetilde{y_j}, p\rangle &= \mathcal{I}[\![x]\!]\langle p\rangle \\ &\approx \mathcal{I}[\![M_j]\!]\langle \widetilde{y_j}, p\rangle \text{ (by Lemma 2.2.13)}\ . \end{aligned}$$

□

Unique solution for $\mathcal{E}_{\mathcal{R}}$. We now prove uniqueness of solutions for $\mathcal{E}_{\mathcal{R}}$. The only delicate requirement is the one on divergence for the syntactic solution. We introduce for this an auxiliary system of equations, $\mathcal{E}'_{\mathcal{R}}$, that extends $\mathcal{E}_{\mathcal{R}}$ (in the sense of Definition 1.1.13). The syntactic solutions of $\mathcal{E}'_{\mathcal{R}}$ have no τ-transition, and hence trivially satisfy the requirement for uniqueness of solutions. We then rely on Lemma 1.1.14 to deduce the same property for $\mathcal{E}_{\mathcal{R}}$.

Like the original system $\mathcal{E}_{\mathcal{R}}$, so the new one $\mathcal{E}'_{\mathcal{R}}$ is defined by inspection of the pairs in $\mathcal{R}$. In $\mathcal{E}'_{\mathcal{R}}$, however, a pair in $\mathcal{R}$ may sometimes yield more than one equation.

As above, we consider $(M, N) \in \mathcal{R}$ with $\widetilde{y} = \mathsf{fv}(M, N)$.

1. When $M \Uparrow$ and $N \Uparrow$, the equation is

   $$X_{M,N} = (\widetilde{y}, p)\ \mathbf{0}\ .$$

2. When $M \Downarrow V$ and $N \Downarrow V'$, we introduce a new equation variable $X^{\mathcal{V}}_{V,V'}$ and a new equation. This allows us, in the following step (3), to perform some optimisations. The equation is

$$X_{M,N} = (\widetilde{y}, p)\ \overline{p}(z).\, X^{\mathcal{V}}_{V,V'}\langle z, \widetilde{y}'\rangle \ ,$$

and we have, accordingly, the two following additional equations corresponding to the cases where values are functions or variables:

$$\begin{aligned} X^{\mathcal{V}}_{\lambda x.M',\lambda x.N'} &= (z, \widetilde{y})\ !z(x,q).\, X_{M',N'}\langle \widetilde{y}', q\rangle \\ X^{\mathcal{V}}_{x,x} &= (z,x)\ z \triangleright x \ . \end{aligned}$$

3. When $M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C_{\mathrm{e}}[xV']$, we refer to $X^{\mathcal{V}}_{V,V'}$, instead of $X_{V,V'}$, so to remove all initial reductions in the corresponding equation for $\mathcal{E}_{\mathcal{R}}$. The first action thus becomes an output:

$$X_{M,N} = (\widetilde{y}, p)\ \overline{x}(z,q).\, (X^{\mathcal{V}}_{V,V'}\langle z, \widetilde{y}'\rangle \mid q(w).\, X_{C_{\mathrm{e}}[w],C'_{\mathrm{e}}[w]}\langle \widetilde{y}'', p\rangle) \ .$$

We now present Lemmas 2.2.24 and 2.2.25, which are are needed to apply Lemma 1.1.14. (In Lemma 2.2.24, 'extend' is as by Definition 1.1.13.)

Lemma 2.2.24. *The system of equations $\mathcal{E}'_{\mathcal{R}}$ extends the system of equations $\mathcal{E}_{\mathcal{R}}$.*

Proof. The new system $\mathcal{E}'_{\mathcal{R}}$ is obtained from $\mathcal{E}_{\mathcal{R}}$ by modifying the equations and adding new ones. To show that the solutions to the common equations are the same, we use the same laws as for the soundness proof. □

Lemma 2.2.25. *$\mathcal{E}'_{\mathcal{R}}$ has a unique solution.*

Proof. Divergence-freedom for the syntactic solutions of $\mathcal{E}'_{\mathcal{R}}$ holds because in the equations each name (bound or free) can appear either only in inputs or only in outputs. More precisely, in the syntactic solutions, linear names $(p, q, \dots)$ are used exactly once in subject position, and non-linear names $(x, y, w, \dots)$, when used in subject position, are either used exclusively in input or exclusively in output.

As a consequence, since the labelled transition system is ground (names are only replaced by fresh ones), no τ-transition can ever be performed, after any number of visible actions. Further, $\mathcal{E}'_{\mathcal{R}}$ is guarded. Hence we can apply Theorem 1.4.11. □

We can remark that a more direct proof of Lemma 2.2.25 would have been possible, by reasoning coinductively over the η-eager normal-form bisimulation defining the system of equations.

Proposition 2.2.26 (Completeness for $\leftrightarrows$)**.** *$M \leftrightarrows N$ implies $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$, for any $M, N \in \Lambda$.*

Proof. Consider an eager normal-form bisimulation $\mathcal{R}$, and the corresponding systems of equations $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{E}'_{\mathcal{R}}$. Lemmas 2.2.25 and 2.2.24 allow us to apply Lemma 1.1.14 and deduce that $\mathcal{E}_{\mathcal{R}}$ has a unique solution. By Lemma 2.2.23, $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of $\mathcal{E}_{\mathcal{R}}$. Thus, from $M\ \mathcal{R}\ N$, we deduce $(\widetilde{y})\ \mathcal{I}[\![M]\!] \approx (\widetilde{y})\ \mathcal{I}[\![N]\!]$, where $\widetilde{y} = \mathsf{fv}(M, N)$. Hence also $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$. □

2.2.7.2 Completeness for $\leftrightarrows_\eta$, and full abstraction

Completeness for $\leftrightarrows_\eta$. The proof for $\leftrightarrows$ is extended to $\leftrightarrows_\eta$, maintaining its structure. We highlight the main differences. The systems of equations are given in full in Appendix D.1.3.

We enrich $\mathcal{E}_\mathcal{R}$ with the equations corresponding to the two additional clauses of $\leftrightarrows_\eta$ (Definition 2.2.6). When $M \Downarrow x$ and $N \Downarrow \lambda z.\, N'$, where $N' \leftrightarrows_\eta xz$, we proceed as in case 4 of the definition of $\mathcal{E}_\mathcal{R}$, given that $N \leftrightarrows_\eta \lambda z.\, ((\lambda w.\, C_{\mathrm{e}}[w])(xV))$. The equation is:

$$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\lambda z.\, \big((\lambda w.\, X_{w,C_{\mathrm{e}}[w]})\ (x\ X_{z,V})\big)]\!]\ .$$

We proceed likewise for the symmetric case.

In the 'optimised equations' defining $\mathcal{E}'_\mathcal{R}$, we add the following equation (relating values), as well as its symmetric counterpart:

$$X^{\mathcal{V}}_{x,\lambda z.N'} = (y_0, \widetilde{y})\ !y_0(z,q).\, \overline{x}(z',q').\, (X^{\mathcal{V}}_{z,V}\langle z', \widetilde{y}'\rangle \mid q'(w).\, X_{w,C_{\mathrm{e}}[w]}\langle \widetilde{y}'', q\rangle)\ .$$

As above, $\mathcal{E}'_\mathcal{R}$ is used to prove uniqueness of solutions for $\mathcal{E}_\mathcal{R}$. In order to prove that $\mathcal{R}$ yields solutions of $\mathcal{E}_\mathcal{R}$, we need to show that the encoding validates η-expansions for variables:

Lemma 2.2.27. $\mathcal{I}[\![\lambda y.\, xy]\!] \approx \mathcal{I}[\![x]\!]$.

Proof. Using standard algebraic laws and the definition of links. □

Finally, we prove that $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of $\mathcal{E}_\mathcal{R}$. Two additional cases are to be considered :

- If $M \Downarrow x$ and $N \Downarrow \lambda y.\, N'$, then:

$$\begin{aligned} E_{M,N}[\mathcal{I}[\![\mathcal{R}_1]\!]]\langle \widetilde{y}, p\rangle &= \overline{p}(z).\, !z(y,q).\, \mathcal{I}[\![(\lambda w.\, w)(xy)]\!]\langle q\rangle \\ &\approx \overline{p}(z).\, !z(y,q).\, \mathcal{I}[\![xy]\!]\langle q\rangle \text{ (by Lemma 2.2.22)} \\ &= \mathcal{I}[\![\lambda y.\, xy]\!]\langle q\rangle \\ &\approx \mathcal{I}[\![x]\!]\langle p\rangle \text{ (by Lemma 2.2.27)} \\ &\approx \mathcal{I}[\![M]\!]\langle \widetilde{y}, p\rangle \text{ (by Lemma 2.2.13)}\ . \end{aligned}$$

- If $M \Downarrow \lambda y.\, M'$, $M' \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow x$, then:

$$\begin{aligned} E_{M,N}[\mathcal{I}[\![\mathcal{R}_1]\!]]\langle \widetilde{y}, p\rangle &= \overline{p}(z).\, !z(y,q).\, \mathcal{I}[\![(\lambda w.\, C_{\mathrm{e}}[w])(xV)]\!]\langle q\rangle \\ &\approx \overline{p}(z).\, !z(y,q).\, \mathcal{I}[\![C_{\mathrm{e}}[xV]]\!]\langle q\rangle \text{ (by Lemma 2.2.22)} \\ &= \mathcal{I}[\![\lambda y.\, C_{\mathrm{e}}[xV]]\!]\langle q\rangle \\ &\approx \mathcal{I}[\![M]\!]\langle \widetilde{y}, p\rangle \text{ (by Lemma 2.2.13)}\ . \end{aligned}$$

Given the previous results, we can reason as for the proof of Proposition 2.2.26 to establish completeness.

Proposition 2.2.28 (Completeness for $\leftrightarrows_\eta$)**.** *For any M, N in Λ, $M \leftrightarrows_\eta N$ implies $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$.*

Combining Propositions 2.2.21 and 2.2.28, and Theorem 1.4.10 we derive Full Abstraction for $\leftrightarrows_\eta$ with respect to barbed congruence.

Theorem 2.2.29 (Full Abstraction for $\leftrightarrows_\eta$)**.** *For any M, N in Λ, we have $M \leftrightarrows_\eta N$ iff $\mathcal{I}[\![M]\!] \simeq^{\mathrm{I}\pi} \mathcal{I}[\![N]\!]$.*

Remark 2.2.30 (Unique solutions versus up-to techniques)**.** For Milner's encoding of call-by-name λ-calculus, the completeness part of the full abstraction result with respect to Levy-Longo Trees [San00] relies on *up-to techniques for bisimilarity*. Precisely, given a relation $\mathcal{R}$ on λ-terms that represents a tree bisimulation, one shows that the π-calculus encoding of $\mathcal{R}$ is a π-calculus bisimulation *up-to context and expansion*.

In the up-to technique, expansion is used to manipulate the derivatives of two transitions so to bring up a common context. Such up-to technique is not powerful enough for the call-by-value encoding and the Eager Trees, because some of the required transformations would violate expansion (i.e., they would require to replace a term by a 'less efficient' one). An example of this is law (2.1) (in the proof of Lemma 2.2.23), that would have to be applied from right to left so to implement the branching in clause (2) of Definition 2.2.3: in this case, the common context is a context with two holes, corresponding to the encoding of $(\lambda z.\,[\cdot_1])(x[\cdot_2])$.

The use of the technique of unique solution of equations allows us to overcome the problem: law (2.1) and similar laws that introduce 'inefficiencies' can be used (and they are indeed used, in various places), as long as they do not produce new divergences.

2.2.8 Encoding into ALπ

Full abstraction with respect to η-Eager-Tree equality also holds for Milner's simplest encoding, namely $\mathcal{V}$ (Section 2.2.2), provided that the target language of the encoding is taken to be ALπ. The adoption of ALπ implicitly allows us to control capabilities, avoiding violations of laws such as Law (5).

2.2.8.1 The Local π-calculus

We present here the results which make it possible for us to apply the unique-solution technique to ALπ. The main idea is to exploit a characterisation of barbed congruence as ground bisimilarity, from [MS04]. However, to obtain this, ground bisimilarity has to be set on top of a non-standard transition system, specialised to ALπ. The Labelled Transition System (LTS) is produced by the rules in Figure 2.5; these modify ordinary transitions (the $\xrightarrow{\mu}$ relation) by adding *static links* $a \blacktriangleright b$, which are abbreviations defined thus:

$$a \blacktriangleright b \stackrel{\text{def}}{=} !a(\widetilde{x}).\,\overline{b}\langle\widetilde{x}\rangle \quad .$$

$$\frac{P \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} P' \qquad \widetilde{p} \cap \mathsf{fn}(P) = \emptyset}{P \overset{\boldsymbol{\nu}\widetilde{p}\,\overline{a}\langle\widetilde{p}\rangle}{\longmapsto} (\widetilde{p} \blacktriangleright \widetilde{b} \mid P')} \qquad \frac{P \xrightarrow{a(\widetilde{b})} P'}{P \overset{a(\widetilde{b})}{\longmapsto} P'} \qquad \frac{P \xrightarrow{\tau} P'}{P \overset{\tau}{\mapsto} P'}$$

Figure 2.5: The modified labelled transition system for ALπ

(We call them *static* links, following the terminology in [MS04], so to distinguish them from the links $a \triangleright b$ used in Iπ, whose definition makes use of recursive process definitions — static links only need replication.)

Notations for the ordinary LTS ($\xrightarrow{\mu}$) are transported onto the new LTS ($\overset{\mu}{\mapsto}$), yielding, e.g., transitions $\overset{\mu}{\Mapsto}$ and $\overset{\hat{\mu}}{\Mapsto}$.

We write $\overset{\leftrightarrow}{\approx}$ for (ground) bisimilarity on the new LTS (defined as $\approx$ in Definition 0.2.2, but using the new LTS in place of the ordinary one). Barbed congruence in ALπ, $\simeq^{\mathrm{AL}\pi}$, is defined as by Definition 1.4.2 (on τ-transitions, which are the only transitions needed to define $\simeq^{\mathrm{AL}\pi}$, the new LTS and the original one coincide).

We present the definition of asynchronous (ground) bisimilarity, which is used in [MS04] to derive a characterisation of barbed congruence (asynchrony is needed because the calculus is asynchronous, and barbed congruence observes only output actions).

Definition 2.2.31 (Asynchronous bisimilarity)**.** Asynchronous bisimilarity, written $\overset{\leftrightarrow}{\approx}_{\mathrm{a}}$, is the largest symmetric relation $\mathcal{R}$ such that $P\mathcal{R}Q$ implies

- if $P \overset{\mu}{\mapsto} P'$ and μ is not an input, then there is Q' s.t. $Q \overset{\hat{\mu}}{\Mapsto} Q'$ and $P'\mathcal{R}Q'$, and
- if $P \overset{a(\widetilde{b})}{\longmapsto} P'$, then either $Q \overset{a(\widetilde{b})}{\Longmapsto} Q'$ and $P' \overset{\leftrightarrow}{\approx}_{\mathrm{a}} Q'$ for some Q', or $Q \Rightarrow Q'$ and $P'\mathcal{R}(Q' \mid \overline{a}\langle\widetilde{b}\rangle)$ for some Q'.

Theorem 2.2.32 ([MS04])**.** *On ALπ processes that are image-finite up to $\overset{\leftrightarrow}{\approx}_{\mathrm{a}}$, relations $\overset{\leftrightarrow}{\approx}_{\mathrm{a}}$ and $\simeq^{\mathrm{AL}\pi}$ coincide.*

To apply our technique of unique solutions of equations it is however convenient to use (synchronous) bisimilarity. The following result allows us to do so:

Theorem 2.2.33. *On ALπ processes that have no free inputs, relations $\overset{\leftrightarrow}{\approx}$ and $\overset{\leftrightarrow}{\approx}_{\mathrm{a}}$ coincide.*

Proof. By construction, $\overset{\leftrightarrow}{\approx} \subseteq \overset{\leftrightarrow}{\approx}_{\mathrm{a}}$.

To show that $\overset{\leftrightarrow}{\approx}_{\mathrm{a}} \subseteq \overset{\leftrightarrow}{\approx}$, we first show that output capability is preserved along transitions: we say that P respects output capability if any free name used in input subject position may not be used in either object position or output subject positions. We show that if P respects output capability and $P \overset{\mu}{\mapsto} P'$, then so does P'.

We reason on the type of the transition $P \xrightarrow{\mu'} P'$ from which $P \overset{P'}{\mapsto}$ is derived (for simplicity, assume monadic actions):

1. if $P \xrightarrow{a(c)} P'$: for simplicity, we consider the transition $a(b).\,Q \overset{a(c)}{\longmapsto} Q\{c/b\}$. c is fresh in Q, and b may not be used in input position in Q. Therefore, c does not appear in input position in $P' = Q\{c/b\}$, and P' respects output capability.

2. if $P \xrightarrow{\tau} P'$: there is no name appearing as both a free input and a free output in P. We assume, for simplicity, $P = \boldsymbol{\nu} a\,(a(b).\,Q \mid \overline{a}c) \overset{\tau}{\mapsto} Q\{c/b\}$. Then, b may not appear in input position in Q, nor does c, as it appears in object position. Hence $P' = Q\{c/b\}$ respects output capability.

3. if $P \xrightarrow{\overline{a}b} P'$: assume for simplicity $P = Q \mid \overline{a}b$, and $P \overset{\overline{a}(c)}{\longmapsto} \boldsymbol{\nu} b\,(c \blacktriangleright b \mid Q)$ $(c \notin \mathsf{fn}(Q))$. The only new input is due to $c \blacktriangleright b$, but c is fresh in Q (and is not used in object position in $c \blacktriangleright b$). Hence $\boldsymbol{\nu} b\,(c \blacktriangleright b \mid Q)$ respects output capability.

4. if $P \xrightarrow{\overline{a}(b)} P'$: same as the previous case.

Thus, we can consider bisimulations containing only processes that respect output capability.

Now, assume $\mathcal{R}$ is an asynchronous bisimulation relation containing only processes that respect output capability. We show that $\mathcal{R}$ is also a synchronous bisimulation relation.

Let $(P, Q) \in \mathcal{R}$. If $P \overset{\mu}{\mapsto} P'$ and μ is not an input action, the synchronous game is played. Otherwise, $P \overset{a(\widetilde{b})}{\longmapsto} P'$. Then either $Q \overset{a(\widetilde{b})}{\Longmapsto} Q'$ for some Q' such that $P' \mathrel{\mathcal{R}} Q'$, and there is nothing to show, or $Q \Mapsto Q'$ for some Q' such that $P' \mathrel{\mathcal{R}} (Q' \mid \overline{a}\langle\widetilde{b}\rangle)$. $Q' \mid \overline{a}\langle b\rangle$ can perform an output on a, thus P' too (by the bisimulation game). Therefore, P has a free input on a, and a free output on a. This is a contradiction. Therefore, only the synchronous game can be played. This is illustrated by the following diagram:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \downarrow a & & \Downarrow = \\ P' & \mathcal{R} & Q' \\ \Downarrow \overline{a} & & \downarrow \overline{a} \end{array}$$

Over processes that respect output capability, asynchronous bisimulation relations are synchronous bisimulation relations, thus $\overset{\leftrightarrows}{\approx}_{\mathrm{a}}$ and $\overset{\leftrightarrows}{\approx}$ coincide. We can conclude: indeed, ALπ processes that have no free input do respect output capability.

□

For any $M \in \Lambda$ and p, process $\mathcal{V}[\![M]\!]\langle p\rangle$ is indeed image-finite up to $\overset{\leftrightarrows}{\approx}_{\mathrm{a}}$ and has no free input, and therefore satisfies the conditions of Theorem 2.2.33. We also exploit the fact that $\overset{\leftrightarrows}{\approx}$ is a congruence relation in ALπ. The property in Theorem 2.2.33 is new | we are not aware of papers in the literature presenting it. It is a consequence of the fact that, under the hypothesis of the theorem, and with a ground transition system, the only input actions in processes that can ever be produced are those emanating from the links, and two tested processes, if bisimilar, must have the same sets of (visible) links.

$M \Uparrow$ and $N \Uparrow$:	$X_{M,N} = (\widetilde{y})\ \mathcal{V}[\![\Omega]\!]$
$M \Downarrow x$ and $N \Downarrow x$:	$X_{M,N} = (\widetilde{y})\ \mathcal{V}[\![x]\!]$
$M \Downarrow \lambda x.\, M'$ and $N \Downarrow \lambda x.\, N'$:	$X_{M,N} = (\widetilde{y})\ \mathcal{V}[\![\lambda x.\, X_{M',N'}]\!]$
$M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C'_{\mathrm{e}}[xV']$:	$X_{M,N} = (\widetilde{y})\ \mathcal{V}[\![(\lambda z.\, X_{C_{\mathrm{e}}[z],C'_{\mathrm{e}}[z]})\ (x\ X_{V,V'})]\!]$
$M \Downarrow x$, $N \Downarrow \lambda z.\, N'$, $N' \Downarrow C_{\mathrm{e}}[xV]$:	$X_{M,N} = (\widetilde{y})\ \mathcal{V}[\![\lambda z.\, \big((\lambda w.\, X_{w,C_{\mathrm{e}}[w]})\ (x\ X_{z,V})\big)]\!]$
$M \Downarrow \lambda z.\, M'$, $M' \Downarrow C_{\mathrm{e}}[xV]$, $N \Downarrow x$:	$X_{M,N} = (\widetilde{y})\ \mathcal{V}[\![\lambda z.\, \big((\lambda w.\, X_{C_{\mathrm{e}}[w],w})\ (x\ X_{V,z})\big)]\!]$

Figure 2.6: System $\mathcal{E}^{\mathbf{L}}_{\mathcal{R}}$ of equations (the last two equations are only needed for $\leftrightarrows_\eta$)

From Theorems 2.2.32 and 2.2.33, we therefore deduce that $\overset{\leftrightarrows}{\approx}$ and $\simeq^{\mathrm{AL}\pi}$ coincide for processes $\mathcal{V}[\![M]\!]p$.

Theorem 2.2.34. *In ALπ, on agents that are image-finite up to $\overset{\leftrightarrows}{\approx}$ and where no free name is used in input, barbed congruence and bisimilarity coincide.*

2.2.8.2 Full abstraction for the encoding into ALπ

We now discuss full abstraction for Milner's encoding $\mathcal{V}[\![\cdot]\!]$, when the target language is ALπ (considered with the modified LTS from Section 2.2.8.1). The proof of is overall very similar to that of Theorem 2.2.29. There are some minor differences because of the modified LTS.

We define two systems of equations for ALπ: $\mathcal{E}^{\mathbf{L}}_{\mathcal{R}}$ (Figure 2.6) and $\mathcal{E}^{\mathbf{L}'}_{\mathcal{R}}$ (Figure 2.7). These are almost the same as $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{E}'_{\mathcal{R}}$, except that the encoding of a term is given by $\mathcal{V}[\![\cdot]\!]$ rather than $\mathcal{I}[\![\cdot]\!]$, with additional static links where appropriate. As above, we write $\widetilde{y}, \widetilde{y}'$ or $\widetilde{y}''$ for the free variables of the terms indexing the corresponding equation variable.

For both systems, we directly give all equations needed to handle $\leftrightarrows_\eta$.

We also recall the extension of the encoding to equation variables:

$$\mathcal{V}[\![X_{M,N}]\!] \stackrel{\text{def}}{=} (p)\ X_{M,N}\langle \widetilde{y}, p\rangle \quad \text{where } \widetilde{y} = \mathsf{fv}(M,N)$$

The main difference with respect to the proofs of Propositions 2.2.26 and 2.2.28 is when proving absence of divergences for the (optimised) system of equations. Indeed, in ALπ's modified transition system, visible transitions may create static links, that could thus produce new reductions (cf. the rule for bound output transitions). Thus one has to show that the added links do not introduce new divergences.

Lemma 2.2.35. *Let P an ALπ process such that P has no divergence according to the ground LTS. Then P has no divergence in the modified LTS for ALπ.*

Proof. When an output occurs, static links are created. Each link is guarded by a replicated input, whose subject is a fresh name. Hence, any synchronisation created by the link has to be preceded by a visible action (the replicated input).

$M \Uparrow$ and $N \Uparrow$:

$$X_{M,N} = (\widetilde{y}, p)\ \mathbf{0}$$

$M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C'_{\mathrm{e}}[xV']$:

$$X_{M,N} = (\widetilde{y}, p)\ (\boldsymbol{\nu} z, q\,)(\overline{x}\langle z, q\rangle \mid X^{\mathcal{V}}_{V,V'}\langle z, \widetilde{y}'\rangle \mid q(w).\, X_{C_{\mathrm{e}}[w],C'_{\mathrm{e}}[w]}\langle \widetilde{y}'', p\rangle)$$

$M \Downarrow V$ and $N \Downarrow V'$:

$$X_{M,N} = (\widetilde{y}, p)\ (\boldsymbol{\nu} y\,)(\overline{p}\langle y\rangle \mid X^{\mathcal{V}}_{v,v'}\langle z, \widetilde{y}'\rangle)$$

$V = x$ and $V' = x$:

$$X^{\mathcal{V}}_{x,x} = (z, x)\ z \blacktriangleright x$$

$V = \lambda x.\, M$ and $V' = \lambda x.\, N$:

$$X^{\mathcal{V}}_{\lambda x.M,\lambda x.N} = (z, \widetilde{y})\ !z(x, q).\, X_{M,N}\langle \widetilde{y}', q\rangle$$

$V = x$, $V' = \lambda z.\, N$, $N \Downarrow C_{\mathrm{e}}[xV'']$:

$$X^{\mathcal{V}}_{x,\lambda z.N} = (y_0, \widetilde{y})\ !y_0(z, q).\, (\boldsymbol{\nu} z', q'\,)(\overline{x}\langle z', q'\rangle \mid X^{\mathcal{V}}_{z,V''}\langle z', \widetilde{y}''\rangle \mid q'(w).\, X_{w,C_{\mathrm{e}}[w]}\langle \widetilde{y}', q\rangle)$$

$V = \lambda z.\, M$, $M \Downarrow C_{\mathrm{e}}[xV'']$, $V' = x$:

$$X^{\mathcal{V}}_{\lambda z.M,x} = (y_0, \widetilde{y})\ !y_0(z, q).\, (\boldsymbol{\nu} z', q'\,)(\overline{x}\langle z', q'\rangle \mid X^{\mathcal{V}}_{V'',z}\langle z', \widetilde{y}''\rangle \mid q'(w).\, X_{C_{\mathrm{e}}[w],w}\langle \widetilde{y}', q\rangle)$$

Figure 2.7: System $\mathcal{E}^{\mathbf{L}'}_{\mathcal{R}}$ of equations (the last two equations are only needed for $\leftrightarrows_\eta$)

Furthermore, only fresh names are transmitted through this synchronisation: in $!a(\widetilde{x}).\, \overline{b}\langle \widetilde{x}\rangle$ the names $\widetilde{x}$ are fresh and then immediately forwarded through b. Therefore, these synchronisations cannot create additional synchronisations, nor can they induce divergences. □

The rest of the proof is very similar to that of Theorem 2.2.29. We finally have

Theorem 2.2.36. *$M \leftrightarrows_\eta N$ iff $\mathcal{V}[\![M]\!] \simeq^{\mathrm{AL}\pi} \mathcal{V}[\![N]\!]$, for any $M, N \in \Lambda$.*

2.2.9 Contextual equivalence and preorders

We have presented full abstraction for η-Eager-Tree equality taking a 'branching' behavioural equivalence, namely barbed congruence, on the π-processes. We show here the same result for contextual equivalence, the most common 'linear' behavioural equivalence. We also extend the results to preorders.

We only discuss the encoding $\mathcal{I}$ into Iπ. Similar results however hold for the encoding $\mathcal{V}$ into ALπ.

2.2.9.1 Contextual relations and traces

Contextual equivalence is defined in the π-calculus analogously to its definition in the λ-calculus (Definition 2.2.2); thus, with respect to barbed congruence, the bisimulation game

on reduction is dropped. Since we wish to handle preorders, we also introduce the *contextual preorder*.

Definition 2.2.37. Two Iπ agents A, B are in the *contextual preorder*, written $A \lesssim_{\mathrm{ct}}^{\mathrm{I}\pi} B$, if $C[A] \Downarrow_a$ implies $C[B] \Downarrow_a$, for all contexts C. They are *contextually equivalent*, written $A \simeq_{\mathrm{ct}}^{\mathrm{I}\pi} B$, if both $A \lesssim_{\mathrm{ct}}^{\mathrm{I}\pi} B$ and $B \lesssim_{\mathrm{ct}}^{\mathrm{I}\pi} A$ hold.

To manage contextual preorder and equivalence in proofs, we exploit characterisations of them as trace inclusion and equivalence.

Theorem 2.2.38. *On Iπ processes, relation $\lesssim_{\mathrm{ct}}^{\mathrm{I}\pi}$ coincides with $\preceq_{\mathrm{tr}}$, and relation $\simeq_{\mathrm{ct}}^{\mathrm{I}\pi}$ coincides with $\approx_{\mathrm{tr}}$.*

We adapt the results for preorders (from the abstract formulation, Section 1.2.3) to Iπ. As the proofs are once again very similar, the details are deferred to Appendix C.2.

Lemma 2.2.39 (Pre-fixed points, $\preceq_{\mathrm{tr}}$)**.** *Let $\mathcal{E}$ be a system of equations, and $\widetilde{\mathrm{K}}_{\mathcal{E}}$ its syntactic solution. If $\widetilde{F}$ is a pre-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{\mathrm{K}}_{\mathcal{E}} \preceq_{\mathrm{tr}} \widetilde{F}$.*

Proof. Consider a finite trace s of $\mathrm{K}_{\widetilde{E},i}\langle\widetilde{a}\rangle$. As it is finite, there must be an n such that s is a trace of $E_i^n\langle\widetilde{a}\rangle$, hence s is also a trace of $E_i^n[\widetilde{F}]\langle\widetilde{a}\rangle$. From $\widetilde{E}[\widetilde{F}] \preceq_{\mathrm{tr}} \widetilde{F}$, by congruence (which holds as a consequence of Theorem 2.2.38), it follows that $E_i^{n+1}[\widetilde{F}] \preceq_{\mathrm{tr}} E_i^n[F_i]$, hence also $E_i^{n+1}[\widetilde{F}] \preceq_{\mathrm{tr}} F_i$. Hence, s is a trace of $F_i\langle\widetilde{a}\rangle$. □

Lemma 2.2.40 (Post-fixed points, $\preceq_{\mathrm{tr}}$)**.** *Let $\mathcal{E}$ be a guarded system of equations, and $\widetilde{\mathrm{K}}_{\mathcal{E}}$ its syntactic solution. Suppose $\widetilde{\mathrm{K}}_{\mathcal{E}}$ has no divergence. If $\widetilde{F}$ is a post-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{\mathrm{K}}_{\mathcal{E}}$.*

From Theorem 2.2.42, we immediatly get the following corollary.

Corollary 2.2.41 (Unique solution for trace equivalence)**.** *In Iπ, a weakly guarded system of equations whose syntactic solution does not diverge has a unique solution for $\approx_{\mathrm{tr}}$.*

Proof. Suppose $\widetilde{F} \approx_{\mathrm{tr}} \widetilde{E}[\widetilde{F}]$ and $\widetilde{G} \approx_{\mathrm{tr}} \widetilde{E}[\widetilde{G}]$; this implies $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{G}$ and $\widetilde{G} \preceq_{\mathrm{tr}} \widetilde{F}$, by applying Theorem 2.2.42 twice. Hence $\widetilde{F} \approx_{\mathrm{tr}} \widetilde{G}$. □

Theorem 2.2.42. *Suppose that $\mathcal{E}$ is a guarded system of equations with a divergence-free syntactic solution.*

If $\widetilde{F}$ (resp. $\widetilde{G}$) is a pre-fixed point (resp. post-fixed point) for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{G}$.

This result is the equivalence for Iπ of Theorem 1.2.27, for the abstract formulation; it follows rather directly from Lemmas 2.2.39 and 2.2.40.

We can also extend Lemma 1.1.14 to preorders. Given a preorder relation $\mathcal{S}$, two systems of equations $\mathcal{E}$ and $\mathcal{E}'$, we say that $\mathcal{E}'$ *extends* $\mathcal{E}$ *with respect to* $\mathcal{S}$ if there exists a fixed set of indices J such that:

1. any pre-fixed point of $\mathcal{E}$ for $\mathcal{S}$ can be obtained from a pre-fixed point of $\mathcal{E}'$ (for $\mathcal{S}$) by removing the components corresponding to indices in J;

2. the same as (1) with post-fixed points in place of pre-fixed points.

Lemma 2.2.43. *Consider two systems of equations $\mathcal{E}'$ and $\mathcal{E}$ where $\mathcal{E}'$ extends $\mathcal{E}$ with respect to $\preceq_{\mathrm{tr}}$. Furthermore, suppose $\mathcal{E}'$ is guarded and has a divergence-free syntactic solution. If $\widetilde{F}$ is a pre-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, and $\widetilde{G}$ a post-fixed point, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{G}$.*

2.2.9.2 Full Abstraction

The preorder on λ-terms induced by the contextual preorder is *η-eager normal-form similarity*, written $\leq_\eta$. It is obtained by imposing that $M \leq_\eta N$ for all N, whenever M is divergent. Thus, with respect to the bisimilarity relation $\leftrightarrows_\eta$, we only have to change clause (1) of Definition 2.2.3, by requiring only M to be divergent. (The bisimilarity $\leftrightarrows_\eta$ is then the intersection of $\leq_\eta$ and its converse $\geq_\eta$.)

Definition 2.2.44 (η-eager normal-form simulation)**.** A relation $\mathcal{R}$ between λ-terms is an η-eager normal-form simulation if, whenever $M\mathcal{R}N$, one of the following holds:

1. M diverges;

2. $M \Longrightarrow C_{\mathrm{e}}[xV]$ and $N \Longrightarrow C'_{\mathrm{e}}[xV']$ for some x, V, V', C_{e} and C'_{e} such that $V\mathcal{R}V'$ and $C_{\mathrm{e}}[z]\mathcal{R}C'_{\mathrm{e}}[z]$ for some z fresh in $C_{\mathrm{e}}, C'_{\mathrm{e}}$;

3. $M \Longrightarrow \lambda x.\, M'$ and $N \Longrightarrow \lambda x.\, N'$ for some x, M', N' such that $M'\mathcal{R}N'$;

4. $M \Longrightarrow x$ and $N \Longrightarrow x$ for some x;

5. $M \Longrightarrow x$ and $N \Longrightarrow \lambda z.\, C_{\mathrm{e}}[xV]$ for some x, z, V and C_{e} such that $z\mathcal{R}V$ and $y\mathcal{R}C_{\mathrm{e}}[y]$ for some y fresh in C_{e};

6. $N \Longrightarrow x$ and $M \Longrightarrow \lambda z.\, C_{\mathrm{e}}[xV]$ for some x, z, V and C_{e} such that $V\mathcal{R}z$ and $C_{\mathrm{e}}[y]\mathcal{R}y$ for some y fresh in C_{e}.

η-eager normal form similarity is the largest η-eager normal-form simulation.

Theorem 2.2.45 (Full abstraction on preorders)**.** *For any $M, N \in \Lambda$, we have $M \leq_\eta N$ iff $\mathcal{I}[\![M]\!] \lesssim_{\mathrm{ct}}^{\mathrm{I}\pi} \mathcal{I}[\![N]\!]$.*

The structure of the proofs is similar to that for bisimilarity, using however Theorem 2.2.42. We discuss the main aspects of the soundness and the completeness.

Soundness means that $\mathcal{I}[\![M]\!] \preceq_{\mathrm{tr}} \mathcal{I}[\![N]\!]$ implies $M \leq_\eta N$. The proof follows the same lines as the proof from Section 2.2.4: we define the relation $\mathcal{R} := \{(M, N) \mid \mathcal{I}[\![M]\!] \preceq_{\mathrm{tr}} \mathcal{I}[\![N]\!]\}$, and show that it is an η-eager normal-form simulation. The proof carries over similarly, using the counterpart for trace inclusion of Lemmas 2.2.15 , 2.2.16, 2.2.17, 2.2.18 and 2.2.20.

To establish completeness, we consider an η-eager normal-form simulation $\mathcal{R}$. We define a system of equations $\mathcal{E}_{\mathcal{R}}$ as in Section 2.2.7. The only notable difference in the definition of the equations is in the case where $M\mathcal{R}N$, M diverges and N has an eager normal-form. In this case, we use the following equation instead:

$$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\Omega]\!]\ . \tag{2.2}$$

As in Section 2.2.7, we define a system of guarded equations $\mathcal{E}'_{\mathcal{R}}$ whose syntactic solutions do not diverge. In doing this, equation (2.2) is replaced with $X_{M,N} = (\widetilde{y}, p)\ \mathbf{0}$. We can then rely on Lemma 2.2.43 to use unique solution for preorders (Theorem 2.2.42) with $\mathcal{E}_{\mathcal{R}}$ instead of $\mathcal{E}'_{\mathcal{R}}$.

Defining $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ as previously, we need to prove

$$\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!] \preceq_{\mathrm{tr}} \widetilde{E_{\mathcal{R}}}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]] \quad \text{and} \quad \widetilde{E_{\mathcal{R}}}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]] \preceq_{\mathrm{tr}} \mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]\ .$$

The former result is established along the lines of the analogous result in Section 2.2.7: indeed, $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ is a solution of $\mathcal{E}_{\mathcal{R}}$ for $\approx$, and $\approx_{\mathrm{tr}}$ is coarser than $\approx$.

For the latter, the only difference is due to equation (2.2), when $M\mathcal{R}N$, and M diverges but N does not. In that case, we have to prove that $\mathcal{I}[\![\Omega]\!] \preceq_{\mathrm{tr}} \mathcal{I}[\![N]\!]$, which follows easily because the only trace of $\mathcal{I}[\![\Omega]\!]$ is the empty one, hence $\mathcal{I}[\![\Omega]\!]\langle p\rangle \preceq_{\mathrm{tr}} P$ for any P.

We obtain full abstraction for contextual equivalence as an immediate corollary.

Corollary 2.2.46 (Full abstraction for $\simeq^{\mathrm{I}\pi}_{\mathrm{ct}}$)**.** *For any M, N in Λ, $M \leftrightarrows_\eta N$ iff $\mathcal{I}[\![M]\!] \simeq^{\mathrm{I}\pi}_{\mathrm{ct}} \mathcal{I}[\![N]\!]$.*

Chapter 3

Conclusion

3.1 The unique-solution proof technique

We have revisited the proof technique of unique solution of equations, more precisely a version of this proof technique due to Roscoe [Ros92, Ros97]. This technique played a central role in the theory of Hoare's CSP [Hoa85] in which it originated, and similar though weaker versions of this technique are used in verification [GM14]. As a technique for coinductive and operational equivalences it was first proposed by Milner; however, apart from its use in examples from Milner's original book on CCS [Mil89], it was mostly ignored, and saw little to no use in the π-calculus or in concurrent languages with higher-order features, until recently Sangiorgi illustrated the relationship between unique-solution and up-to-context techniques [San15], that are central enhancements of the bisimulation proof technique.

The main contribution of this thesis, in the field of π-calculus and coinductive equivalences, has been to dust off and rediscover this technique for coinductive equivalences and operational semantics: we proved the soundness of one of its most powerful incarnations, for different equivalences, compared it to other techniques, and applied to non-trivial examples, including an open problem.

3.1.1 The importance of up to context

Another goal of this work was to illustrate the relationship of this technique with up-to techniques, relationship that we tried to explore and formalize. In the Prologue, we exposed how central up-to techniques are to the theory of bisimulations, and more specifically the importance of 'up to context' techniques, but also the difficulties when applying such techniques, and when proving them to be sound. We gave multiple examples from the literature of situations where up-to-context is either unsound, impossible to combine with other techniques, or its soundness is still an open problem. In all of these cases, unique solution of equations provides an interesting alternative.

Another problem with up-to techniques for weak equivalences is the unsoundness of up to bisimilarity (or at least of up to context and bisimilarity, see Section 0.4.2.3). A usual fix is to consider preorders that are 'sensitive to efficiency', e.g. that control the relative number

of internal steps that the two processes playing the bisimulation game may perform; such preorders include expansion (Definition 0.3.11) and contraction (Definition 0.6.1). However these preorders limit, sometimes severely, the application of the technique (an example is given by the call-by-value encoding in π, as discussed in Remark 2.2.30 and below). The only existing technique that proposes a solution to these limitations is Pous' 'up to context, transitivity and bisimilarity' technique (defined in Section 1.1.4.1), one of the most powerful forms of enhancements of the bisimulation proof method. We showed that our technique is at least as powerful; furthermore, the up-to technique is arguably more complex, both in its definition and its application. We believe that also the converse holds (Pous' technique is as powerful as our technique), though possibly with some additional side conditions (the non-divergence hypothesis of our technique has to hold for some contexts only, while termination needs a full context closure for Pous' up-to technique), and only so for CCS. We leave a detailed analysis of this comparison, which seems non-trivial, for future work. This result also begs the question of the completeness of our technique with respect to up-to techniques for other settings: while the proof in Section 1.1.4.1 is language-independent (under certain assumptions) and should apply to most first-order synchronous settings, the relationship between bisimulation enhancements and unique solution of equation theorems appears to be weaker in name-passing calculi. In this respect, the goal of the work on unique solution of equations was also to provide a way of better understanding up-to techniques and to shed light into the conditions for their soundness, as discussed hereafter.

Lastly, up-to techniques have been analysed in an abstract setting using lattice theory [Pou16] and category theory [BPPR17, RBR13]. It could be interesting to do the same for the unique-solution techniques, to study their connections with up-to techniques, and to understand which equivalences can be handled (possibly using, or refining, the abstract formulation presented in Section 1.2.1).

3.1.2 Problems with the π-calculus and Higher-Order

Open Problem 0.4.13 (the need for closure under substitutions in up to context for the asynchronous π-calculus) is a long-standing problem, and a crucial missing piece of the theory of up-to techniques in name-passing calculi. Surprisingly, our unique solution technique, despite these strong similarities with up-to context techniques, does not require the closure under substitutions. However, it is currently unclear how to formally relate bisimulation enhancements and 'unique solution of equations' in name-passing calculi, particularly if those calculi have constraints such as asynchrony (Aπ, ALπ), type constraints (i/o types), or other. Even if there is no formal correspondence between up-to-context techniques and unique-solution techniques in this setting, the latter might act as a useful substitute for the former in most cases.

We directly established our main unique-solution theorem and their variants for multiple subcalculi of the π-calculus. This illustrates the robustness of our proof, that scales with few to no modifications. A more systematic approach, however, would save a direct proof in each and every case, giving unique-solution theorems for free (similar to Theorems 1.2.16 and 1.2.21). A possible direction to investigate is that of first-order encodings [MPS14]: a

framework to encode an LTS with name-binding as a first-order CCS-like LTS. The abstract results from Section 1.2 might then directly apply to such settings, as long as the encodings respect certain properties.

The Higher-order π-calculus. The case of HOπ is original in many regards. First, because of process-passing, enforcing non-divergence for regular bisimulations is unreasonable. On the other hand, the contraction approach does not seem applicable because of the substitutivity problems for normal bisimilarity (that also prevent the use of up-to-context techniques), which would also show up with contraction, as discussed at the end of Section 1.4.2.2 (the problems with the the expansion preorder and the contraction preorder are similar).

Similar problems arise for other equivalences: for context bisimulation (discussed in Section 1.4.2.2), the clauses of the bisimulation involve universal quantifications on terms supplied by the external environment, the current conditions on divergence might be too restrictive; thus, the treatment of divergence might need refinements.

Other divergence-related problems are discussed in Example 1.4.27: some forms of divergence deserve further investigation, in order to refine the notion of innocuous divergence. Indeed, the transmission of a process, which can be executed, can generate infinite sequences of communication steps. Intuitively, this kind of divergence is not intrinsic to the original process performing the input (one may choose not to execute the transmitted process). We need to understand how this fits in our theory.

The λ-calculus. The unique-solution technique should scale for fined-grained equivalences of the λ-calculus, such as eager normal form bisimilarity or open bisimilarity, if only by virtue of Milner's fully abstract encodings to Aπ and Iπ (Chapter 2). On the other hand for contextual equivalence, problems similar to HOπ arise: divergences are the main observable, and even non-innocuous divergences restrict the techniques to terms that might never diverge. Defining a notion of contraction preorder seems really difficult. There are, however, interesting uses of up-to-context in the λ-calculus, such as Biernacki et al. [BLP18]. We would like to see if similar enhancements can also be applied to higher-order processes and to the unique-solution technique.

3.1.3 Using the technique

Unique-solution techniques have been studied in many different settings [Mil89, Hoa85, Ros92, GM14]. Our originality has been to propose a generic proof, that scales to multiple languages and most importantly different equivalences; to propose an abstract setting formalizing the generic aspect of the proof, and pinpointing precisely the required hypotheses; and proposing an improvement of the technique, which is specific to the operational nature of our approach.

In comparison with the enhancements of the bisimulation proof method, the main drawback of the techniques we developed is the presence of a semantic condition, involving di-

vergence: the unfoldings of the equations should not produce divergences, or only produce innocuous divergences. A syntactic condition for this has been proposed (e.g., variants of Lemma 1.1.12). Various techniques for checking divergence in concurrent calculi exist in the literature, including type-based techniques [YBH04, DHS10, DS06]. However, in general divergence is undecidable, and therefore, the check may sometimes be unfeasible. Nevertheless, the equations that one writes for proofs usually involve forms of 'normalised' processes, and as such they are divergence-free (or at most, contain only innocuous divergences). Surprinsingly, this is confirmed in the case of the encodings of the λ-calculus in the π-calculus, even though the main observable in the λ-calculus is precisely divergences. More experiments are needed to validate this claim or to understand how limiting this problem is.

Several studies in functional programming and type theory rely on type-based methods to ensure that coinductive definitions are productive, i.e., do not give rise to partially defined functions (in order to preserve logical consistency) [Nak00, AM13]. Understanding whether these approaches can be adapted to analyse divergences and innocuous divergences in systems of equations is also a topic for future investigations.

3.2 Open call-by-value in the π-calculus

3.2.1 Milner's open problem

The other main contribution of this thesis is our work on Milner's open problem of Full Abstraction for the encoding of call-by-vaue in the π-calculus, for which we proposed a solution.

First we have shown that some expected equalities for open terms fail under Milner's encoding. We have considered two ways for overcoming this issue: rectifying the encodings (precisely, avoiding free outputs); or restricting the target language to ALπ, so to control the capabilities of exported names. We have proved that, in both cases, the equivalence induced is Eager-Tree equality, modulo η (i.e., Lassen's η-eager normal-form bisimulation). For controlling capabilities, we have used ALπ. Another possibility would have been to use a type system (such as i/o types [PS96], which can indeed enforce the behavioural constraints of ALπ). In this case however, the technique of unique solution of equations needs to be extended to typed calculi. We leave this investigation for future work.

On the π-calculus side, we studied the question for both contextual equivalence and barbed congruence (the most common 'linear' and 'branching' equivalences), and for the contextual preorder; for the latter, we had to introduce a notion of preorder on Lassen's trees (η-eager normal-form simulation), quite naturally derived from η-eager normal-form bisimulation.

For our encodings we have used the polyadic π-calculus; Milner's original paper [Mil90a] used the monadic calculus (the polyadic π-calculus makes the encoding easier to read; it had not been introduced at the time of [Mil90a]). We believe that polyadicity does not affect our results (the possibility of autoconcurrency breaks full abstraction of the encoding of the polyadic π-calculus into the monadic one, but autoconcurrency does not appear in the

encoding of λ-terms).

Other encodings In the call-by-value strategy we have followed, the function is reduced before the argument in an application. Our results can be adapted to the case in which the argument runs first, changing the definition of evaluation contexts. The parallel call-by-value, in which function and argument can run in parallel (considered in [Mil92]), appears more delicate, as we cannot rely on the usual notion of evaluation context; furthermore, the encodings of two diverging λ-terms evaluating in parallel can be different: for instance, the π-calculus distinguishes between the term $(xv)\Omega$ and the term $(yv)\Omega$ (because the calls to instantiate variables x and y, respectively, are observable). A full abstraction theorem would thus necessitate an equivalence that discriminates between pure diverging terms; such behavioural equivalences are uncommon, and to our knowledge, there is no such equivalence for the parallel *call-by-value* λ-calculus in the literature.

There are also other encodings of λ-calculus strategies in the π-calculus: for *call-by-need*, for instance [SW01], as well as a strong *call-by-name* encoding [SX14, MS04], that is fully abstract for Böhm Tree equality. There are also encodings of Higher-Order calculi (HOπ) in the π-calculus [San93a], and encodings of the λ-calculus in other Higher-Order calculi, such as HOcore [BBL$^+$17], a small but expressive process calculus. It would be interesting to investigate, in future work, to what extent our technique could apply to those settings and be used to prove full abstraction results.

Completeness proofs and the unique-solution techniques. It is plausible that the proof based on the unique-solution technique could be turned into an up-to bisimulation. However this would, at the very least, necessitate the use of an optimized encoding: indeed, take for instance the encoding of $C_{\mathrm{e}}[xV]$; it contains several administrative reductions, that only occur later in the encoding of $(\lambda z.\, C_{\mathrm{e}}[z])(xV)$, hence it would not be possible to use the expansion pre-order. This is different from a direct bisimulation proof, as these transformations cannot be performed through an existing up-to technique. This is one of the very rare cases where there is no obvious way to obtain a bisimulation up to expansion and context from a unique-solution proof.

We leave for future investigation a formal study of this question, including if and how our completeness results can be derived using the latter techniques or other (new) up-to techniques, in a similar way as they are used in the completeness proofs with respect to Levy-Longo Trees and Böhm Trees for the *call-by-name* encodings. We have discussed the problems with call-by-value in Remark 2.2.30.

In any case, regardless of whether other proof techniques may be used, for this specific problem the unique-solution technique appears to provide an elegant and natural framework to carry out the proofs.

3.2.2 Building models with the π-calculus

Behavioural equivalences for open call-by-value. If the choice of an adequate proof technique constituted an important obstacle to Full Abstraction for Milner's call-by-value encoding, another obstacle is that of the equivalence for which it is fully abstract: indeed, equivalences for open call-by-value (call-by-value for open terms) are not as well understood as equivalences for call-by-name, and standard equivalences from the theory of call-by-name have no equivalence for call-by-value.

Lassen had introduced Eager Trees as the call-by-value analogous of Levy-Longo trees. Our results would tend to confirm his claim, as it was known that for call-by-name, the tree equalities induced by π-calculus encodings are Levy-Longo Trees. For strong call-by-name, the equivalence induced is Böhm Trees equality [SX14]; however there is no Böhm Tree equivalent for strong call-by-value, and it is as well unclear how to define a π-calculus encoding for strong call-by-value. The obstacles impeding the definition of an equivalence for strong call-by-value are the same we find in the case of open call-by-value, and as such are related to the π-calculus encodings. In fact, the very notions of reduction and normal forms are not clear for call-by-value, as terms can be stuck before reaching an actual value ($I(xV)$ does not reduce to xV). Lassen's eager normal bisimulations had been proposed as a first solution to this problem; we discuss some other solutions that have been proposed.

Related work on open and strong call-by-value. Open and strong call-by-value have been studied in [AG16], where the focus is on operational properties of λ-terms; behavioural equivalences are not considered. The paper describes the fireball calculus. As in our case, the calculus is weak (no reduction is possible under a lambda), call-by-value, but, contrarily to our setting, $xV_1 \cdots V_k$ is treated as a value, i.e., β-reduction can be triggered when the argument has this shape.

A different origin for this calculus is the work by Grégoire and Leroy about the implementation of Coq [GL02], which uses strong reduction. There, the approach is to rely on a evaluator for the weak version of the calculus, and progress by levels: first reduce at top level, then go under lambda, hence opening the terms, and so on. This way open terms arise, and this is why they study a weak strategy dealing with open terms.

An extensive presentation of call-by-value, including denotational models, is Ronchi della Rocca and Paolini's book [RP04].

For future investigation, it would be interesting to understand the relationship between the π-calculus encodings and these theories of call-by-value, and whether these reduction strategies can be faithfully represented in the π-calculus. We discuss possible directions to use the π-calculus to represent models that are fully abstract with respect to to contextual equivalence. This full abstraction property is one of the goals of previous variants of strong call-by-value.

Models from Milner's encoding. The standard behavioural equivalence in the λ-calculus is contextual equivalence. Encodings into the π-calculus (be it for call-by-name or call-by-value) break contextual equivalence, at least without specific tweaks, because π-calculus

contexts are richer than those in the (pure) λ-calculus. We tried to understand how far beyond contextual equivalence the discriminating power of the π-calculus brings us, for call-by-value. The opposite approach is to restrict the set of 'legal' π-contexts so to remain faithful to contextual equivalence. This approach has been followed, for call-by-name, and using type systems, in [BHY01, TY18].

Lassen et al. showed in [SL09] that eager normal form bisimilarity coincides with contextual equivalence for a (typed) variant of the μ-calculus with references; in other words, given the results we established, the discriminating power of π-contexts is that of λ-contexts with side-effects and Call/CC. This leads to two interesting research directions:

1. Looking to establish Full Abstraction for contextual equivalence, for an encoding that translates references and exceptions to the π-calculus. We know how to translate references [Wal95], we are not aware of π-calculus translations of Call/CC, which is used in Lassen's calculus. It is unclear whether using exceptions instead of Call/CC would make the situation easier to handle.

2. Trying to recover Full Abstraction for contextual equivalence by diminishing the discriminating power of π contexts: either by modifying the encoding, restricting the language (for instance, by considering a subcalculus), or making bisimilarity on π processes coarser.

π-calculus and games. Game semantics models of the π-calculus are trace-based, interactive, and usually fully abstract models, first introduced in the typed setting of PCF by Hyland & Ong [HO00]. They rely, in their more concrete instantiations, on the description of an operational semantics for partial λ-terms [GT12], and the use of global criteria on traces (such as '*innocence*' [HO00]).

Game semantics accounts of untyped call-by-name λ-calculus exist [GFH99], and they allow us to derive both Böhm Trees [KNO03] and Levy-Longo Trees [OG04]. Game semantics of untyped λ-calculus was also studied in [KNO99, KNO02]. Thus, for untyped call-by-name, game models and the π-calculus encodings agree. In fact, game semantics appear to be closely related to π-calculus representations of the λ-calculus: this relationship has been first formally established in [HO95], where innocent traces from the game model of PCF are represented using π-processes. This relationship, in the setting of call-by-name, has been further explored using session types [CY19].

In this regard, our contribution on full abstraction for Milner's encoding would tend to validate both the idea that Eager Trees are the call-by-value counterpart of Levy-Longo Trees, and as such correspond to game models, and the idea that the π-calculus and games are strongly related, and that π-calculus encodings might be a very explicit and concrete way to describe game models – before doing any sort of quotient of the model based on 'bad' traces.

However, while game semantics accounts of typed call-by-value do exist, e.g., [AM97, HY99], we know of no game semantics accounts for the untyped case (albeit methods to extract a game semantics for an untyped calculus from its typed counterpart do exist, as described in [GFH99, McC00]).

Given this correspondence, it would be interesting to see whether our contributions can help defining a game semantics approach to untyped call-by- value, based on Eager Trees, and whether the relationship between π-calculus and Eager Trees we studied could help to establish similar relationships in game semantics.

To go even further, this new work on π-calculus encodings might be an additional tool to understand the general relation between the already-existing encodings and the already-existing game models; in turn, this suggests to try and generalize this correspondence between games and the π-calculus, and describe systematically game models as π encodings.

In conclusion, we believe the π-calculus might be the adequate tool to describe and *implement* 'concrete' – i.e. operational, tractable – game models; π should be able to provide powerful proof techniques (based on 'bisimulations up to' or 'unique solution of equations', for instance) to study these models.

To be continued...

Bibliography

[Abr87] S. Abramsky. The lazy λ-calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1987.

[AC15] Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155, 2015.

[ACS98] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.

[AFV01] Luca Aceto, Wan Fokkink, and Chris Verhoef. *Handbook of Process Algebra (J.A. Bergstra and A. Ponse and S.A. Smolka, editors)*, chapter Structural operational semantics. Elsevier Science, 2001.

[AG16] Beniamino Accattoli and Giulio Guerrieri. Open call-by-value (extended version). *CoRR*, abs/1609.00322, 2016.

[AM97] Samson Abramsky and Guy McCusker. Call-by-value games. In *Proceedings of, CSL '97, Annual Conference of the EACSL, Selected Papers*, pages 1–17, 1997.

[AM13] Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 197–208, 2013.

[Bar84] H.P. Barendregt. *The lambda calculus: its syntax and semantics*. Studies in logic and the foundations of mathematics. North-Holland, 1984.

[BBL+17] Malgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, Piotr Polesiuk, Damien Pous, and Alan Schmitt. Fully Abstract Encodings of λ-Calculus in HOcore through Abstract Machines. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

[BBR10] Jos C.M. Baeten, Twan Basten, and Michel A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2010.

[BC03] Vincent D. Blondel and Vincent Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory Comput. Syst.*, 36(3):231–245, 2003.

[BCI+16] Marc Brockschmidt, Byron Cook, Samin Ishtiaq, Heidy Khlaaf, and Nir Piterman. T2: temporal property verification. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 387–393, 2016.

[BGZ09] Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(6):1191–1222, 2009.

[BHR84] Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[BHY01] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the pi-calculus. In *TLCA*, pages 29–45, 2001.

[BIM88] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 229–239, 1988.

[Blo95] Bard Bloom. Structural operational semantics for weak bisimulations. *Theor. Comput. Sci.*, 146(1&2):25–68, 1995.

[BLP18] Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Proving soundness of extensional normal-form bisimilarities. *Electr. Notes Theor. Comput. Sci.*, 336:41–56, 2018.

[Bou00] Gérard Boudol. On the semantics of the call-by-name CPS transform. *Theor. Comput. Sci.*, 234(1-2):309–321, 2000.

[BP13] Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 457–468, 2013.

[BP15a] Manuel Bodirsky and Michael Pinsker. Topological birkhoff. *Transactions of the American Mathematical Society*, 367(4):2527–2549, 2015.

[BP15b] Filippo Bonchi and Damien Pous. Hacking nondeterminism with induction and coinduction. *Commun. ACM*, 58(2):87–95, 2015.

[BPP17] Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Reconstructing the topology of clones. *Transactions of the American Mathematical Society*, 369(5):3707–3740, 2017.

[BPPR17] Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. A general account of coinduction up-to. *Acta Inf.*, 54(2):127–190, 2017.

[BR84] Stephen D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA*, pages 281–305, 1984.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, New York, NY, USA, 1990.

[Coq93] Thierry Coquand. Infinite objects in type theory. In *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24-28, 1993, Selected Papers*, pages 62–78, 1993.

[CY19] Simon Castellan and Nobuko Yoshida. Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side. *PACMPL*, 3(POPL):27:1–27:29, 2019.

[DHL06] Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'06, pages 273–284, 2006.

[DHS10] Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. Termination in impure concurrent languages. In *CONCUR 2010 - Concurrency Theory, 21th International Conference*, pages 328–342, 2010.

[DHS17] Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Divergence and Unique Solution of Equations. In *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85, pages 11:1–11:16, 2017.

[DHS18] Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Eager functions as processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 364–373, 2018.

[DHS19] Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Divergence and unique solution of equations. *Logical Methods in Computer Science*, 15(3), 2019.

[DHS20] Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Towards 'up to context' reasoning about higher-order processes. *Theor. Comput. Sci.*, 807:154–168, 2020.

[dS85] Robert de Simone. Higher-level synchronising devices in meije-sccs. *Theor. Comput. Sci.*, 37:245–267, 1985.

[DS06] Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. *Inf. Comput.*, 204(7):1045–1082, 2006.

[Dur17] Adrien Durier. Divergence and unique solution of equations in an abstract setting, Coq formal proof. Available at `https://github.com/adurier/uniquesolution/`, 2017.

[FvG16] Wan Fokkink and Rob J. van Glabbeek. Divide and congruence II: delay and weak bisimilarity. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 778–787. ACM, 2016.

[GFH99] Pietro Di Gianantonio, Gianluca Franco, and Furio Honsell. Game semantics for untyped lambda beta eta-calculus. In *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications*, TLCA '99, pages 114–128, London, UK, UK, 1999. Springer-Verlag.

[Gim94] Eduardo Giménez. Codifying guarded definitions with recursive schemes. In *Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6-10, 1994, Selected Papers*, pages 39–59, 1994.

[GL02] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002.*, pages 235–246, 2002.

[GM14] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[GT12] Dan R. Ghica and Nikos Tzevelekos. A system-level game semantics. In Ulrich Berger and Michael W. Mislove, editors, *Proceedings of the 28th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2012, Bath, UK, June 6-9, 2012*, volume 286 of *Electronic Notes in Theoretical Computer Science*, pages 191–211. Elsevier, 2012.

[GV92] Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comput.*, 100(2):202–260, 1992.

[HNDV13] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 193–206, 2013.

[HO95] J. M. E. Hyland and C.-H. Luke Ong. Pi-calculus, dialogue games and PCF. In *Proceedings of FPCA 1995*, pages 96–107. ACM, 1995.

[HO00] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP07] Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, 2007.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and Lambda-Calculus*. Cambridge University Press, 1986.

[HY99] Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1-2):393–456, 1999.

[KNO99] Andrew D. Ker, Hanno Nickau, and C.-H. Luke Ong. A universal innocent game model for the böhm tree lambda theory. In *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, pages 405–419, 1999.

[KNO02] Andrew D. Ker, Hanno Nickau, and C.-H. Luke Ong. Innocent game models of untyped lambda-calculus. *Theor. Comput. Sci.*, 272(1-2):247–292, 2002.

[KNO03] Andrew D. Ker, Hanno Nickau, and C.-H. Luke Ong. Adapting innocent game models for the böhm treelambda-theory. *Theor. Comput. Sci.*, 308(1-3):333–366, 2003.

[KW06] Vassileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proc. POPL'06*, pages 141–152, 2006.

[Las98a] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, 1998.

[Las98b] Søren B. Lassen. Relational reasoning about contexts. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 91–135. Cambridge University Press, 1998.

[Las05] Søren B. Lassen. Eager normal form bisimulation. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 345–354, 2005.

[Ler] Xavier Leroy. CompCert C verified compiler. `http://compcert.inria.fr/`.

[Lév75] Jean-Jacques Lévy. An algebraic interpretation of the lambda beta - calculus and a labeled lambda - calculus. In *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, March 25-27, 1975*, pages 147–165, 1975.

[LL05] Søren B. Lassen and Paul Blain Levy. Eager normal form bisimulation. In *In Proc. 20th Annual IEEE Symposium on Logic in Computer Science*, pages 345–354. IEEE Computer Society, 2005.

[Lon83] Giuseppe Longo. Set-theoretical models of λ-calculus: theories, expansions, isomorphisms. *Annals of Pure and Applied Logic*, 24(2):153 – 188, 1983.

[McC00] Guy McCusker. Games and full abstraction for FPC. *Inf. Comput.*, 160(1-2):1–61, 2000.

[Mil89] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[Mil90a] Robin Milner. Functions as processes. Research Report RR-1154, INRIA, 1990.

[Mil90b] Robin Milner. Functions as processes. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, pages 167–180, 1990.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil93] Robin Milner. The polyadic π-calculus: a tutorial. In *Logic and algebra of specification*, volume 94 of *NATO ASI Series (Series F: Computer & Systems Sciences)*, pages 203–246. Springer, 1993.

[Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[MN05] Massimo Merro and Francesco Zappa Nardelli. Behavioral theory for mobile ambients. *J. ACM*, 52(6):961–1023, 2005.

[MPS14] Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 93–108, 2014.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[MRG07] Mohammad Reza Mousavi, Michel A. Reniers, and Jan Friso Groote. SOS formats and meta-theory: 20 years after. *Theor. Comput. Sci.*, 373(3):238–272, 2007.

[MS92] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 685–695, 1992.

[MS04] Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004.

[Nak00] Hiroshi Nakano. A modality for recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 255–266, 2000.

[OG04] C.-H. Luke Ong and Pietro Di Gianantonio. Games characterizing levy-longo trees. *Theor. Comput. Sci.*, 312(1):121–142, 2004.

[OY16] Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 568–581, 2016.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[Pou07] Damien Pous. Complete lattices and up-to techniques. In *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007, Proceedings*, pages 351–366, 2007.

[Pou08] Damien Pous. *Up to techniques for bisimulations*. PhD thesis, ENS Lyon, February 2008.

[Pou16] Damien Pous. Coinduction all the way up. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 307–316, 2016.

[Pou17] Damien Pous. private communication, 2017.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[PS11] Damien Pous and Davide Sangiorgi. *Advanced Topics in Bisimulation and Coinduction (D. Sangiorgi and J. Rutten editors)*, chapter Enhancements of the coinductive proof method. Cambridge University Press, 2011.

[RBR13] Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Coalgebraic bisimulation-up-to. In *SOFSEM 2013: Theory and Practice of Computer Science, 39th International Conference on Current Trends in Theory and Practice of Computer Science*, pages 369–381, 2013.

[Ros92] A. W. Roscoe. An alternative order for the failures model. *J. Log. Comput.*, 2(5):557–577, 1992.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[Ros10] A. W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010.

[RP04] Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus - A Metamodel for Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[San93a] Davide Sangiorgi. *Expressing mobility in process algebras : first-order and higher-order paradigms*. PhD thesis, University of Edinburgh, UK, 1993.

[San93b] Davide Sangiorgi. An investigation into functions as processes. In *Proc. of MFPS'93*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 1993.

[San94] Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Inf. Comput.*, 111(1):120–153, 1994.

[San95] Davide Sangiorgi. On the proof method for bisimulation (extended abstract). In *Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Prague, Czech Republic, August 28 - September 1, 1995, Proceedings*, pages 479–488, 1995.

[San96a] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Inf. Comput.*, 131(2):141–178, 1996.

[San96b] Davide Sangiorgi. pi-calculus, internal mobility, and agent-passing calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996.

[San98] Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

[San00] Davide Sangiorgi. Lazy functions and mobile processes. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[San11] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, New York, NY, USA, 2011.

[San15] Davide Sangiorgi. Equations, contractions, and unique solutions. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*, pages 421–432, 2015.

[SB81] Hanamantagouda P Sankappanavar and Stanley Burris. *A course in universal algebra*, volume 78. 1981.

[SKS11] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5:1–5:69, 2011.

[SL09] Kristian Støvring and Søren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, pages 329–375, 2009.

[SM92] Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In *CONCUR '92, Third International Conference on Concurrency Theory, Stony Brook, NY, USA, August 24-27, 1992, Proceedings*, pages 32–46, 1992.

[SR11] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[SX14] Davide Sangiorgi and Xian Xu. Trees from functions as processes. In *25th International Conference, CONCUR 2014, Rome, Italy*, LNCS, pages 78–92. Springer Verlag, 2014.

[Sze86] Ágnes Szendrei. Clones in universal algebra. *Les presses de L'universite de Montreal*, 1986.

[TCP12] Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 346–360, 2012.

[Tea] The Coq Development Team. Coq proof assistant. https://coq.inria.fr/.

[TY18] Bernardo Toninho and Nobuko Yoshida. On polymorphic sessions and functions - A tale of two (fully abstract) encodings. In *Proc. of ESOP 2018*, volume 10801 of *Lecture Notes in Computer Science*, pages 827–855. Springer, 2018.

[UP02] Irek Ulidowski and Iain C. C. Phillips. Ordered SOS process languages for branching and eager bisimulations. *Inf. Comput.*, 178(1):180–213, 2002.

[vG90] Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, pages 278–297, 1990.

[vG93] Rob J. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, pages 66–81, 1993.

[vG05] Rob J. van Glabbeek. On cool congruence formats for weak bisimulations. In *Theoretical Aspects of Computing - ICTAC 2005, Second International Colloquium*, pages 318–333, 2005.

[vG11] Rob J. van Glabbeek. On cool congruence formats for weak bisimulations. *Theor. Comput. Sci.*, 412(28):3283–3302, 2011.

[Wal95] David Walker. Objects in the pi-calculus. *Inf. Comput.*, 116(2):253–271, 1995.

[YBH04] Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong Normalisation in the Pi-Calculus. *Information and Computation*, 191(2):145–202, 2004.

Appendix A

List of relation symbols

A.1 List of symbols for behavioural relations

The following table summarises the notations used for the equivalences and preorders.

Equivalences for process calculi.

$\approx$	(weak) bisimilarity	(Definition 0.2.2, Prologue - Section 0.2.1)
$\sim$	strong bisimilarity	(Definition 0.2.1, Prologue - Section 0.2.1)
$\preceq$	expansion	(Definition 0.3.11, Prologue - Section 0.3.3.3)
$\succeq_c$	contraction	(Definition 0.6.1, Prologue - Section 0.6)
$\approx_{\mathrm{tr}}$	trace equivalence	(Definition 0.2.11, Prologue - Section 0.2.3)
$\preceq_{\mathrm{tr}}$	trace inclusion	(Definition 0.2.11, Prologue - Section 0.2.3)
$\simeq^{\bullet}$	barbed congruence in CCS	(Definition 0.2.8, Prologue - Section 0.2.2)
$\simeq^{\mathcal{L}}$	barbed congruence in $\mathcal{L}$	(Definition 1.4.2, Prologue - Section 1.4.1.2)
$\simeq^{\mathcal{L}}_{\mathrm{ct}}$	contextual equivalence in $\mathcal{L}$	(Definition 2.2.37, Section 2.2.9.1)
$\lesssim^{\mathrm{I}\pi}_{\mathrm{ct}}$	contextual preorder in Iπ	(Definition 2.2.37, Section 2.2.9.1)
$\approx^{\mathrm{gr}}$	ground bisimilarity, ALπ special LTS	(Section 2.2.8.1)

where $\mathcal{L}$ is supposed to be a subcalculus of π. In the paper we have considered Iπ and ALπ.

Equivalences for the λ-calculus.

$\leftrightarrows$	eager normal-form bisimilarity	(Definition 2.2.3, Section 2.2.1.1)
$\leftrightarrows_\eta$	η-eager normal-form bisimilarity	(Definition 2.2.6, Section 2.2.1.1)
$\leq_\eta$	η-eager normal-form similarity	(Definition 2.2.44, Section 2.2.9.2)
$\simeq^{\Lambda}_{\mathrm{ct}}$	contextual equivalence in Λ	(Definition 2.2.2, Section 2.2.1)

Appendix B

Up-to-context techniques

B.1 Non-compatibility of up to context in Aπ

This appendix consists in a proof that the substitution closure function

$$\mathfrak{S} : \mathcal{R} \mapsto \{(\sigma(P), \sigma(Q)) \mid P \mathcal{R} Q \text{ and } \sigma \text{ is any substitution}\}$$

is not compatible in the asynchronous π-calculus, and that neither is the context closure function $\mathcal{C}$ (see Section 0.4.4 and reference [PS11] for more details).

This proof is due to Damien Pous [Pou17], exchanged in a personal communication; to our knowledge, such a proof as never been published, this is why we reproduce it here, with his authorisation.

We recall that $\mathbf{b}$ is the monotone function on the lattice of binary relations on processes defined as :

$$\begin{aligned}\mathbf{b}(\mathcal{R}) \stackrel{\triangle}{=} \{(P,Q) \mid\ & \forall \mu, \\ & \forall P', P \xrightarrow{\mu} P' \text{ implies there exists } Q', Q \xrightarrow{\mu} Q' \text{ and } P' \mathcal{R} Q' \\ & \forall Q', Q \xrightarrow{\mu} Q' \text{ implies there exists } P', P \xrightarrow{\mu} P' \text{ and } P' \mathcal{R} Q'\}\end{aligned}$$

As per usual, we use this function to characterize strong bisimilarity as a largest fixed-point: $\nu\mathbf{b} = \sim$ [San11, Mil89, SR11, Pou16]. We define the successive approximations of strong bisimilarity, written $\sim_i$ for an index $i \in \mathbb{N}$, as follows:

- $\sim_0 \stackrel{\triangle}{=} \{(P,Q) \mid \text{ for all processes } P \text{ and } Q\}$ is the full relation
- $\sim_{i+1} \stackrel{\triangle}{=} \mathbf{b}(\sim_i)$

We use the following characterisation of compatibility [SR11]: f is below a compatible function iff for all i, $f(\sim_i) \leq \sim_i$. We show that the substitution σ such that $\sigma(a) = b$ and is otherwise the identity is not compatible, by showing $\sigma(\sim_2) \not\leq \sim_2$.

First, remark that $a.\,\overline{b} \mid \overline{b} \sim_2 a \mid \overline{b}$. However, σ breaks this equivalence: $\sigma(a.\,\overline{b} \mid \overline{b}) = a.\,\overline{a} \mid \overline{a} \xrightarrow{\tau} \overline{a}$, however $\sigma(a \mid \overline{b}) = a \mid \overline{a} \xrightarrow{\tau} \mathbf{0}$; thus $\sigma(a.\,\overline{b} \mid \overline{b}) \not\sim_2 \sigma(a \mid \overline{b})$. Thus $\sigma(\sim_2) \not\leq \sim_2$, and $\mathfrak{S}$ is not compatible (nor below any compatible function).

We now use the same example to show that $\mathcal{C}$ is not compatible either. Assume $P = a.\,(a.\,\overline{b} \mid \overline{b}) \mid \overline{b}$ and $Q = a.\,(a \mid \overline{b}) \mid \overline{b}$. We have that $P \sim_4 Q$. However, writing C for the context $\boldsymbol{\nu} c\,(c(a).\,\cdot \mid \overline{c}b)$, we have that $C[P] \not\sim_4 C[Q]$ (arguments are similar to above). Therefore up-to input context is not compatible (nor below any compatible function).

Appendix C

Proofs of variants of the main theorem

C.1 Unique solution proofs in HOπ

This Appendix contains detailed proofs for the unique solution theorem in HOπ (Theorems 1.4.24 and 1.4.25).

In analysing such transitions, there are several aspects which differ between CCS and HOπ. First, we rely on the notions of reducts (Definition 1.4.23), which, as explained above, have to be tailored to HOπ. Moreover, input and output transitions yield abstractions and concretions, which need to be instantiated. This shows up in several places below, like in the following definition, in clauses 3 and 4, when we need to refer to processes in order to impose a condition involving $\approx$.

Definition C.1.1. A system of equations $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$ *protects its solutions* if, for all solution $\widetilde{F}$, and for all $\mathbf{P}' \in \mathtt{red}_\omega(\widetilde{\mathbf{F}})$, the following hold:

1. if $\mathbf{P}'[\widetilde{F}] \Longrightarrow Q$ for some Q, then there exists $\mathbf{P}''$ and n such that $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}''$ and $\mathbf{P}''[\widetilde{F}] \approx Q$.

2. if $\mathbf{P}'[\widetilde{F}] \stackrel{\ell}{\Rightarrow} Q$ for some Q, then there exists $\mathbf{P}''$ and n such that $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{\ell}{\Rightarrow} \mathbf{P}''$ and $\mathbf{P}''[\widetilde{F}] \approx Q$.

3. if $\mathbf{P}'[\widetilde{F}] \stackrel{a}{\Rightarrow} F_0$, for some a and some F, then there exists $\mathbf{F}$ and n such that $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{a}{\Rightarrow} \mathbf{F}_0$ and $(\mathbf{F}_0 \circ \mathtt{Tr}_m)[\widetilde{F}] \approx F_0 \circ \mathtt{Tr}_m$ (for any m fresh).

4. if $\mathbf{P}'[\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C$, for some a and some C, then there exists $\mathbf{C}$ and n such that $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{\overline{a}}{\Rightarrow} \mathbf{C}$ and $(\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}] \approx \mathbf{C} \bullet \mathtt{Ab}_m$ (for any m fresh).

Intuitively, a system of equations $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$ protect its solutions when transitions emanating from $\mathbf{P}[\widetilde{F}]$, where $\mathbf{P} \in \mathtt{red}_\omega(\widetilde{\mathbf{F}})$ and $\widetilde{F}$ is a solution, can be mimicked by transitions of $\mathbf{P}[\widetilde{\mathbf{F}}^n]$ for some n — i.e., by replacing the solutions with unfoldings of the equations.

Proposition C.1.2. *A system of equations that protects its solutions has a unique solution for* $\approx$.

Proof. We have to show that given two solutions $\widetilde{F}$, $\widetilde{F}'$ of the system of equations $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$, and a fresh name m_0, it holds that $\widetilde{F} \circ \mathtt{Tr}_{m_0} \approx \widetilde{F}' \circ \mathtt{Tr}_{m_0}$.

With these hypotheses, we set $\widetilde{E} = \widetilde{\mathbf{F}} \circ \mathtt{Tr}_{m_0}$, $\widetilde{P} = \widetilde{F} \circ \mathtt{Tr}_{m_0}$ and $\widetilde{P}' = \widetilde{F}' \circ \mathtt{Tr}_{m_0}$; we have that $\widetilde{E}[\widetilde{F}] \approx \widetilde{P}$ and $\widetilde{E}[\widetilde{F}'] \approx \widetilde{P}'$, and, by definition, $\mathtt{red}_\omega(\widetilde{E}) \subset \mathtt{red}_\omega(\widetilde{\mathbf{F}})$.

We prove that the relation

$$\mathcal{R} \quad \triangleq \quad \{(S,T) \mid \exists \mathbf{P}', \text{ s.t. } S \approx \mathbf{P}'[\widetilde{F}],\, T \approx \mathbf{P}'[\widetilde{F}'] \text{ and } \mathbf{P}' \in \mathtt{red}_\omega(\widetilde{E})\}$$

is a bisimulation relation satisfying $\widetilde{P}\mathcal{R}\widetilde{Q}$.

We consider $(S,T) \in \mathcal{R}$, that is, $S \approx \mathbf{E}'[\widetilde{F}]$ and $T \approx \mathbf{E}'[\widetilde{F}]'$, for some $\mathbf{P}' \in \mathtt{red}_\omega(\widetilde{E})$. We reason about the possible transitions from S.

1. Suppose $S \xrightarrow{\tau} S'$. We deduce:

   - $\mathbf{E}'[\widetilde{F}] \Longrightarrow S'' \approx S'$ (by bisimilarity).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}''$ and $\mathbf{P}''[\widetilde{F}] \approx S''$ for some n, $\mathbf{P}''$ ($\widetilde{\mathbf{F}}$ protects its solutions).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] = \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] \Longrightarrow \mathbf{P}''[\widetilde{F}']$ (by Lemmas 1.4.19 and 1.4.21(1)).
   - $T \approx \mathbf{E}'[\widetilde{F}'] \approx \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] = \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']]$ ($T \approx \mathbf{P}'[\widetilde{F}']$ and $\widetilde{F}' \approx \widetilde{\mathbf{F}}^n[\widetilde{F}']$, because $\widetilde{F}'$ is solution of $\widetilde{\mathbf{F}}$).
   - $T \Longrightarrow T' \approx \mathbf{P}''[\widetilde{F}']$ (by bisimilarity, from $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] \Longrightarrow \mathbf{P}''[\widetilde{F}']$).

   This situation can be depicted on the following diagram:

$$\begin{array}{ccccccccccccc}
S & \approx & \mathbf{E}'[\widetilde{F}] & \approx & \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}]] & & \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] & \approx & \mathbf{E}'[\widetilde{F}'] & \approx & T \\
{\scriptstyle\tau}\downarrow & & \Downarrow & & \Downarrow & = & \Downarrow & & & & \Downarrow \\
S' & \approx & S'' & \approx & \mathbf{P}''[\widetilde{F}] & & \mathbf{P}''[\widetilde{F}'] & & \approx & & T'
\end{array}$$

   From $\mathbf{P}' \in \mathtt{red}_\omega(\widetilde{E})$ and $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}''$, we deduce that $\mathbf{P}'' \in \mathtt{red}_\omega(\widetilde{E})$. Finally, $T' \approx \mathbf{P}''[\widetilde{F}']$ and $S' \approx \mathbf{P}''[\widetilde{F}]$ give that $(S,T) \in \mathcal{R}$.

2. Similarly, suppose $S \xrightarrow{\ell} S'$. We deduce:

   - $\mathbf{E}'[\widetilde{F}] \stackrel{\ell}{\Rightarrow} S'' \approx S'$ (by bisimilarity).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{\ell}{\Rightarrow} \mathbf{P}''$ and $\mathbf{P}''[\widetilde{F}] \approx S''$ for some n, $\mathbf{P}''$ ($\widetilde{\mathbf{F}}$ protects its solutions).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] = \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] \stackrel{\ell}{\Rightarrow} \mathbf{P}''[\widetilde{F}']$ (by Lemmas 1.4.19 and 1.4.21(1))
   - $T \approx \mathbf{E}'[\widetilde{F}'] \approx \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] = \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']]$ ($T \approx \mathbf{P}'[\widetilde{F}']$ and $\widetilde{F}' \approx \widetilde{\mathbf{F}}^n[\widetilde{F}']$, because $\widetilde{F}'$ is solution of $\widetilde{\mathbf{F}}$).
   - $T \stackrel{\ell}{\Rightarrow} T' \approx \mathbf{P}''[\widetilde{F}']$ (by bisimilarity, from $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] \stackrel{\ell}{\Rightarrow} \mathbf{P}''[\widetilde{F}']$).

From $\mathbf{P}' \in \mathtt{red}_\omega(\widetilde{E})$ and $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{\ell}{\Rightarrow} \mathbf{P}''$, we deduce that $\mathbf{P}'' \in \mathtt{red}_\omega(\widetilde{E})$. Finally, $T' \approx \mathbf{P}''[\widetilde{F}']$ and $S' \approx \mathbf{P}''[\widetilde{F}]$ give that $(S,T) \in \mathcal{R}$.

3. Suppose now $S \stackrel{a}{\Rightarrow} F$. We deduce:

   - $\mathbf{E}'[\widetilde{F}] \stackrel{a}{\Rightarrow} F'$ and $F \circ \mathtt{Tr}_m \approx F' \circ \mathtt{Tr}_m$ for some m (by bisimilarity).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{a}{\Rightarrow} \mathbf{F}$ and $(\mathbf{F} \circ \mathtt{Tr}_m)[\widetilde{F}] \approx F' \circ \mathtt{Tr}_m$ for some n, $\mathbf{F}$ ($\widetilde{\mathbf{F}}$ protects its solutions).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] = \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] \stackrel{a}{\Rightarrow} \mathbf{F}[\widetilde{F}']$. (by Lemmas 1.4.19 and 1.4.21(1)).
   - $T \approx \mathbf{E}'[\widetilde{F}'] \approx \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] = \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']]$ ($T \approx \mathbf{P}'[\widetilde{F}']$ and $\widetilde{F}' \approx \widetilde{\mathbf{F}}^n[\widetilde{F}']$, because $\widetilde{F}'$ is solution of $\widetilde{\mathbf{F}}$).
   - $T \stackrel{a}{\Rightarrow} F''$ and $F'' \circ \mathtt{Tr}_m \approx \mathbf{F}[\widetilde{F}'] \circ \mathtt{Tr}_m = (\mathbf{F} \circ \mathtt{Tr}_m)[\widetilde{F}']$ for some F'' (by bisimilarity, from $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] \stackrel{a}{\Rightarrow} \mathbf{F}[\widetilde{F}']$).

   From $\mathbf{P}' \in \mathtt{red}_\omega(\widetilde{E})$ and $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{a}{\Rightarrow} \mathbf{F}$, we deduce that $\mathbf{F} \circ \mathtt{Tr}_m \in \mathtt{red}_\omega(\widetilde{E})$. Finally, $F'' \circ \mathtt{Tr}_m \approx (\mathbf{F} \circ \mathtt{Tr}_m)[\widetilde{F}']$ and $F \circ \mathtt{Tr}_m \approx (\mathbf{F} \circ \mathtt{Tr}_m)[\widetilde{F}]$ give that $(F \circ \mathtt{Tr}_m, F'' \circ \mathtt{Tr}_m) \in \mathcal{R}$.

4. Again, suppose now that $S \stackrel{\overline{a}}{\Rightarrow} C$. Again:

   - $\mathbf{E}'[\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C'$ and $C \bullet \mathtt{Ab}_m \approx C' \bullet \mathtt{Ab}_m$ for some m (by bisimilarity).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{\overline{a}}{\Rightarrow} \mathbf{C}$ and $(\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}] \approx C' \bullet \mathtt{Ab}_m$ for some n, $\mathbf{C}$ ($\widetilde{\mathbf{F}}$ protects its solutions).
   - $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] = \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] \stackrel{\overline{a}}{\Rightarrow} \mathbf{C}[\widetilde{F}']$ (by Lemmas 1.4.19 and 1.4.21(1)).
   - $T \approx \mathbf{E}'[\widetilde{F}'] \approx \mathbf{P}'[\widetilde{\mathbf{F}}^n][\widetilde{F}'] = \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']]$ ($T \approx \mathbf{E}'[\widetilde{F}'] \approx \mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']]$, because $\widetilde{F}'$ is solution of $\widetilde{\mathbf{F}}$).
   - $T \stackrel{\overline{a}}{\Rightarrow} C''$ and $C'' \bullet \mathtt{Ab}_m \approx \mathbf{C}[\widetilde{F}'] \bullet \mathtt{Ab}_m = (\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}']$ (by Lemma 1.4.22 and by bisimilarity, from $\mathbf{P}'[\widetilde{\mathbf{F}}^n[\widetilde{F}']] \stackrel{a}{\Rightarrow} \mathbf{F}[\widetilde{F}']$).

   From $\mathbf{P}' \in \mathtt{red}_\omega(\widetilde{E})$ and $\mathbf{P}'[\widetilde{\mathbf{F}}^n] \stackrel{\overline{a}}{\Rightarrow} \mathbf{C}$, we get that $\mathbf{C} \bullet \mathtt{Ab}_m \in \mathtt{red}_\omega(\widetilde{E})$.

   Finally, $C'' \bullet \mathtt{Ab}_m \approx (\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}']$ and $C \bullet \mathtt{Ab}_m \approx (\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}]$ allow us to deduce that $(C \bullet \mathtt{Ab}_m, C'' \bullet \mathtt{Ab}_m) \in \mathcal{R}$.

We reason symmetrically for the actions of T. □

We say that the syntactic solutions of the system $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$ do not diverge if, for all $i \in I$, $K_{\widetilde{\mathbf{F}},i}$ does not diverge.

Theorem 1.4.24 (Unique solution)**.** *A guarded system of equations whose syntactic solutions do not diverge has a unique solution for* $\approx$.

Proof. Given a guarded system of equations $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$ whose syntactic solutions do not diverge, we prove that $\widetilde{\mathbf{F}}$ protects its solutions.

To prove that, we need to consider some $\mathbf{P}_0 \in \mathtt{red}_\omega(\widetilde{\mathbf{F}})$ and some solution $\widetilde{F}$ of $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$. There are four cases to consider, according to the four clauses of Definition C.1.1.

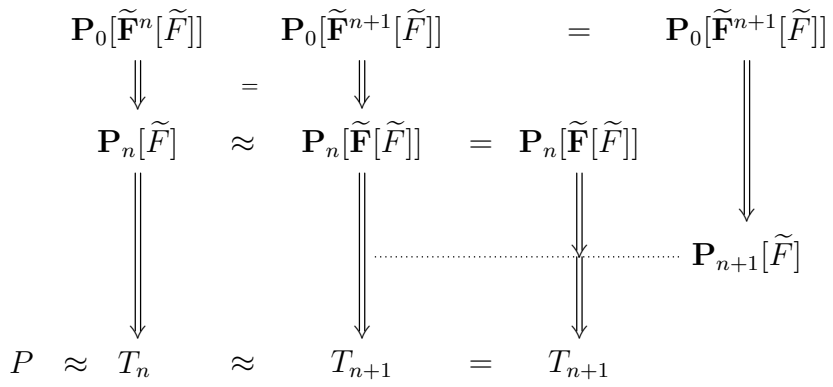


Figure C.1: Recursion: construction of the sequence of transitions $\mathbf{P}_0[\mathbf{F}^n] \Longrightarrow \mathbf{P}_n$

1. Consider a transition $\mathbf{P}_0[\widetilde{F}] \Longrightarrow P$.

   We build a sequence of extended processes $\mathbf{P}_n$ and an increasing sequence of transitions $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}_n$ such that: either this construction stops, yielding a transition $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}_n$, or the construction is infinite, therefore giving a divergence of $\widetilde{K}_{\widetilde{\mathbf{F}}}$.

   We build this sequence so that it additionally satisfies:

   - $\mathbf{P}_n[\widetilde{F}] \Longrightarrow\approx P$,
   - The sequence is strictly increasing: the sequence of transitions $\mathbf{P}_0[\widetilde{\mathbf{F}}^{n+1}] \Longrightarrow \mathbf{P}_n[\widetilde{\mathbf{F}}]$ is a strict prefix of the sequence of transitions $\mathbf{P}_0[\widetilde{\mathbf{F}}^{n+1}] \Longrightarrow \mathbf{P}_{n+1}$ (hence $\mathbf{P}_n[\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_{n+1}$).

   This construction is illustrated by Figure C.2. We initialise with the empty sequence from $\mathbf{P}_0$.

   Then, at step n, we have $\mathbf{P}_0[\widetilde{\mathbf{F}}^n[\widetilde{F}]] \Longrightarrow \mathbf{P}_n[\widetilde{F}] \Longrightarrow T_n$, with $P \approx T_n$.

   - If $\mathbf{P}_n[\widetilde{F}] \Longrightarrow T_n$ is the empty sequence, we stop. We have in this case $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}_n$ and $P \approx \mathbf{P}_n[\widetilde{F}]$.
   - Otherwise (as depicted on Figure C.1), we unfold the equations in $\widetilde{\mathbf{F}}$ one more time. We deduce
     - $\mathbf{P}_0[\widetilde{\mathbf{F}}^{n+1}] = \mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_n[\widetilde{\mathbf{F}}]$ (by Lemmas 1.4.19 and 1.4.21), and
     - $\mathbf{P}_n[\widetilde{\mathbf{F}}[F]] = \mathbf{P}_n[\widetilde{\mathbf{F}}][F] \Longrightarrow T_{n+1} \approx P$ for some T_{n+1} (by Lemma 1.4.19, congruence and bisimilarity).

     If $\mathbf{P}_n[\widetilde{\mathbf{F}}[F]] \Longrightarrow T_{n+1}$ is the empty sequence, we stop as previously. Otherwise, we take $\mathbf{P}_n[\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_{n+1}$ to be the longest prefix sequence of transitions in $\mathbf{P}_n[\widetilde{\mathbf{F}}][F] \Longrightarrow T_{n+1}$ that are instances of transitions from $\mathbf{P}_n[\widetilde{\mathbf{F}}]$ (we remark that as $\mathbf{P}_n[\widetilde{\mathbf{F}}]$ is guarded, by Lemma 1.4.20, this sequence is not empty).

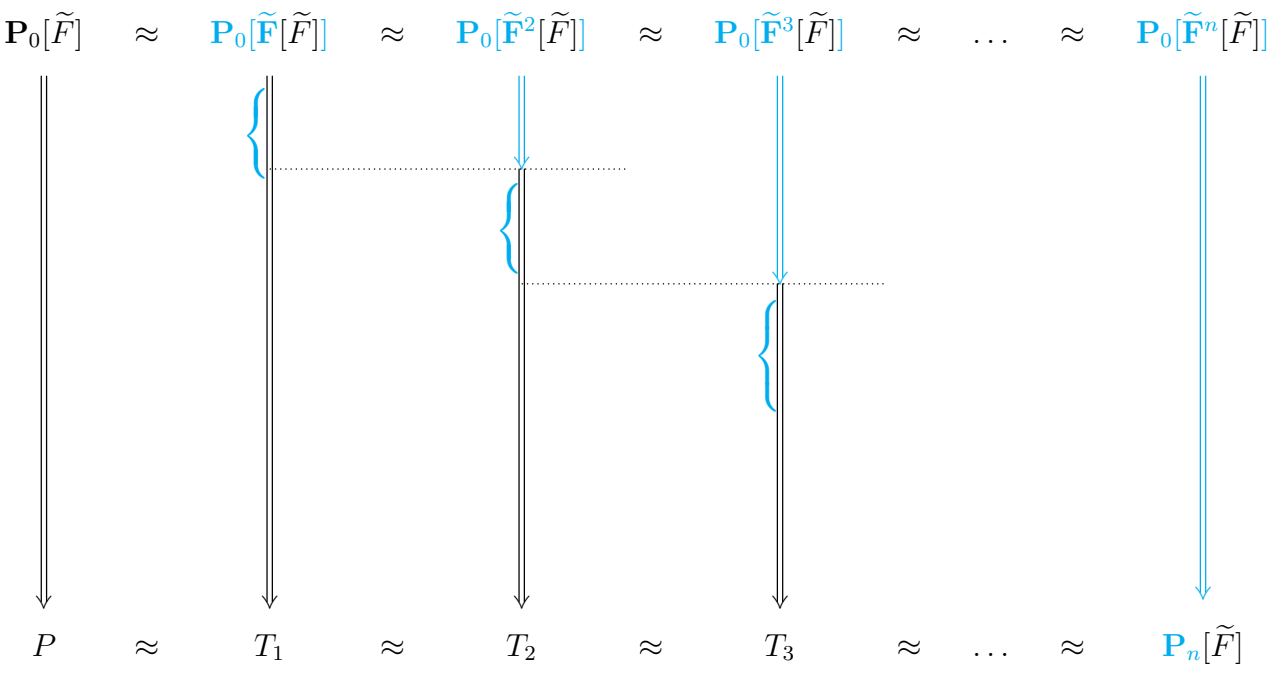


Figure C.2: Construction of the transition

Suppose now that the construction given above never stops. We know that $\mathbf{P}_n[\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_{n+1}$, therefore $\mathbf{P}_n[\widetilde{K}_{\widetilde{\mathbf{F}}}] \Longrightarrow \mathbf{P}_{n+1}[\widetilde{K}_{\widetilde{\mathbf{F}}}]$. This gives an infinite sequence of transitions starting from $\mathbf{P}_0[\widetilde{K}_{\widetilde{\mathbf{F}}}]$: $\mathbf{P}_0[\widetilde{K}_{\widetilde{\mathbf{F}}}] \Longrightarrow \mathbf{P}_1[\widetilde{K}_{\widetilde{\mathbf{F}}}] \Longrightarrow \ldots$. We observe that in the latter sequence, every step involves at least one transition, and moreover, there is no visible action occurring in this infinite sequence. Therefore $\mathbf{P}_0[\widetilde{K}_{\widetilde{\mathbf{F}}}]$ is divergent, which contradicts the hypothesis of the theorem. Hence, the construction does stop, and this concludes the proof.

Before moving on to the next case, we can remark that here, in contrast with the situation in CCS, no τ can follow an action triggered by a prefix in a weak transition: by definition (Section 1.4.2.2), $\stackrel{\mu}{\Longrightarrow}$ stands for $\Longrightarrow$ followed by $\xrightarrow{\mu}$. This means that in the construction given above, (i) we stop as soon as said action is performed, and (ii) we directly build a divergence if the construction never stops, rather than a sequence of transitions leading to a divergence.

2. Now consider a transition $\mathbf{P}_0[\widetilde{F}] \stackrel{\ell}{\Rightarrow} P$. The construction is similar to that of the previous point, for a transition $\Longrightarrow$; we just build the sequence $\mathbf{P}_n$ so that it satisfies $\mathbf{P}_n[\widetilde{F}] \stackrel{\ell}{\Rightarrow}\!\approx P$.

   For the same reason as in the previous case, the construction necessarily stops, otherwise we would observe a divergence of the syntactic solutions.

3. Consider a transition $\mathbf{P}_0[\widetilde{F}] \stackrel{a}{\Rightarrow} F'$. Likewise, we build an increasing sequence of

transitions $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}_n$, such that when the construction stops, it yields a transition $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \stackrel{a}{\Rightarrow} \mathbf{F}'$, for some equation abstraction $\mathbf{F}'$ such that $(\mathbf{F}' \circ \mathtt{Tr}_m)[\widetilde{F}] \approx F' \circ \mathtt{Tr}_m$, for some m fresh. We fix such an m.

The sequence also satisfies:

- $\mathbf{P}_n[\widetilde{F}] \stackrel{a}{\Rightarrow} G_n$ for some G_n, and $G_n \circ \mathtt{Tr}_m \approx (\mathbf{F}' \circ \mathtt{Tr}_m)[\widetilde{F}]$.
- The sequence is strictly increasing.

Again, we initialise with the empty sequence from $\mathbf{P}_0$. Suppose we are at step n. We then have

- $\mathbf{P}_0[\mathbf{F}^n[\widetilde{F}]] \Longrightarrow \mathbf{P}_n[\widetilde{F}] \stackrel{a}{\Rightarrow} G_n$ with $G_n \circ \mathtt{Tr}_m \approx F' \circ \mathtt{Tr}_m$.
- $\mathbf{P}_0[\widetilde{\mathbf{F}}^{n+1}] = \mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_n[\widetilde{\mathbf{F}}]$ (by Lemmas 1.4.19 and 1.4.21).
- $\mathbf{P}_n[\widetilde{\mathbf{F}}[\widetilde{F}]] = \mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{a}{\Rightarrow} G_{n+1}$ for some G_{n+1} such that $G_{n+1} \circ \mathtt{Tr}_m \approx G_n \circ \mathtt{Tr}_m$ (by Lemma 1.4.19, congruence and bisimilarity).

If $\mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{a}{\Rightarrow} G_{n+1}$ is an instance of a transition from $\mathbf{P}_n[\widetilde{\mathbf{F}}]$, meaning $\mathbf{P}_n[\widetilde{\mathbf{F}}] \stackrel{a}{\Rightarrow} \mathbf{F}'$ for some $\mathbf{F}'$ such that $\mathbf{F}'[\widetilde{F}] = G_{n+1}$, we stop. Such a transition $\mathbf{P}_n[\widetilde{\mathbf{F}}] \stackrel{a}{\Rightarrow} \mathbf{F}'$ satisfies indeed the desired properties.

Otherwise, we take $\mathbf{P}_n[\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_{n+1}$ to be the longest prefix sequence of transitions in $\mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{a}{\Rightarrow} G_{n+1}$ that are instances of transitions from $\mathbf{P}_n[\widetilde{\mathbf{F}}]$. It must be a strict prefix, otherwise $\mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{a}{\Rightarrow} G_{n+1}$ would be an instance of a transition from $\mathbf{P}_n[\widetilde{\mathbf{F}}]$; it is indeed a transition $\Longrightarrow$ (as the input a must be the last performed action), and since $\mathbf{P}_n[\widetilde{\mathbf{F}}]$ is guarded (by Lemma 1.4.20), it is not the empty sequence.

Again, the construction necessarily stops, otherwise it gives an infinite sequence of transitions from $\mathbf{P}_0[\widetilde{K}_{\widetilde{\mathbf{F}}}]$, hence a divergence of the syntactic solutions.

4. Consider a transition $\mathbf{P}_0[\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C$. The proof is similar to the previous case: we build an increasing sequence of transitions $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \Longrightarrow \mathbf{P}_n$; when the construction stops, it yields a transition $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \stackrel{\overline{a}}{\Rightarrow} \mathbf{C}$ such that $(\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}] \approx C \bullet \mathtt{Ab}_m$ (m fixed). The sequence is still strictly increasing, and $\mathbf{P}_n[\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C_n$ with $C_n \bullet \mathtt{Ab}_m \approx (\mathbf{C} \bullet \mathtt{Ab}_m)[\widetilde{F}]$.

   At step n, we have

   - $\mathbf{P}_0[\widetilde{\mathbf{F}}^n[\widetilde{F}]] \Longrightarrow \mathbf{P}_n[\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C_n$ with $C_n \bullet \mathtt{Ab}_m \approx C \bullet \mathtt{Ab}_m$.
   - $\mathbf{P}_0[\widetilde{\mathbf{F}}^{n+1}] = \mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_n[\widetilde{\mathbf{F}}]$ (by Lemmas 1.4.19 and 1.4.21).
   - $\mathbf{P}_n[\widetilde{\mathbf{F}}[\widetilde{F}]] = \mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C_{n+1}$ for C_{n+1} such that $C_{n+1} \bullet \mathtt{Ab}_m \approx C_n \bullet \mathtt{Ab}_m$ (by Lemma 1.4.19, congruence and bisimilarity).

   If $\mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C_{n+1}$ is an instance of a transition from $\mathbf{P}_n[\widetilde{\mathbf{F}}]$ we stop. Otherwise, we take $\mathbf{P}_n[\widetilde{\mathbf{F}}] \Longrightarrow \mathbf{P}_{n+1}$ to be the longest strict prefix sequence of transitions in

$\mathbf{P}_n[\widetilde{\mathbf{F}}][\widetilde{F}] \stackrel{a}{\Rightarrow} C_{n+1}$ that are instances transitions from $\mathbf{P}_n[\widetilde{\mathbf{F}}]$. It is not the empty sequence, as by Lemma 1.4.20, $\mathbf{P}_n[\widetilde{\mathbf{F}}]$ is guarded.

Again, the termination argument still applies.

This concludes the proof. □

C.1.1 Innocuous Divergences in HOπ

Theorem 1.4.25 (Unique solution with innocuous divergences)**.** *Let $\widetilde{\mathbf{X}} = \widetilde{\mathbf{F}}$ be a system of guarded equations, and $\widetilde{K}_{\widetilde{\mathbf{F}}}$ be its syntactic solutions. If for any i, all divergences of $K_{\widetilde{\mathbf{F}},i}$ are innocuous, then $\widetilde{\mathbf{F}}$ has a unique solution for $\approx$.*

Proof. We reason like in the proof of Theorem 1.4.24. Cases 2, 3 and 4, corresponding to inputs and outputs, are handled in the same way in the present proof.

The differences arise in case 1, corresponding to a τ transition.

- If at some point the transition $\mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{F}] \stackrel{\tau}{\Rightarrow} T_n$ is an instance of a transition $\mathbf{P}_0[\widetilde{\mathbf{F}}^n] \stackrel{\tau}{\Rightarrow} \mathbf{P}_n$, then the construction can stop.
- If on the other hand the construction never stops, we can build a non innocuous divergence: we can assume that for any n, $\mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{F}] \stackrel{\tau}{\Rightarrow} T_n$ is not an instance of a transition from $\mathbf{P}_0[\widetilde{\mathbf{F}}^n]$ (case 1), and likewise for transitions $\mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{F}] \stackrel{a}{\Rightarrow} F_n$ or $\mathbf{P}_0[\widetilde{\mathbf{F}}^n][\widetilde{F}] \stackrel{\overline{a}}{\Rightarrow} C_n$. This means that in the sequence of transitions leading to the divergence, the constant rule is used at least n times applied to the constants $K_{\widetilde{\mathbf{F}},i}$ for various i. Hence, the divergence we build uses at least n times the constant rule for all n. Therefore, it is not innocuous, which is a contradiction.

□

C.2 Unique solution for contextual relations in Iπ

We provide some details of the proofs of the unique solution techniques for trace preorders and equivalence (Section 2.2.9).

Lemma 2.2.39 (Pre-fixed points, $\preceq_{\mathrm{tr}}$)**.** *Let $\mathcal{E}$ be a system of equations, and $\widetilde{\mathrm{K}}_{\mathcal{E}}$ its syntactic solution. If $\widetilde{F}$ is a pre-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{\mathrm{K}}_{\mathcal{E}} \preceq_{\mathrm{tr}} \widetilde{F}$.*

Proof. Suppose $\{\widetilde{X} = \widetilde{E}\}$ is a guarded system of equations, $\widetilde{\mathrm{K}}_{\widetilde{E}}$ its syntactic solution, and $\widetilde{E}[\widetilde{F}] \preceq_{\mathrm{tr}} \widetilde{F}$. We have to show that $\widetilde{\mathrm{K}}_{\widetilde{E}} \preceq_{\mathrm{tr}} \widetilde{F}$, i.e., for any i and any ground instantiation of the i-th constant, say $\mathrm{K}_{\widetilde{E},i}\langle\widetilde{a}\rangle$, a trace s for $\mathrm{K}_{\widetilde{E},i}\langle\widetilde{a}\rangle$ is also a trace for $F_i\langle\widetilde{a}\rangle$.

As s is finite, for deriving the trace s from $\mathrm{K}_{\widetilde{E},i}\langle\widetilde{a}\rangle$, the constants $\widetilde{\mathrm{K}}_{\widetilde{E}}$ have been unfolded a finite number of times, say n. This means that s is also a trace for $(E_i^n[\widetilde{G}])\langle\widetilde{a}\rangle$, for any $\widetilde{G}$; in particular also for $(E_i^n[\widetilde{F}])\langle\widetilde{a}\rangle$.

Since $\widetilde{F}$ is a pre-fixed point (that is, $\widetilde{E}[\widetilde{F}] \preceq_{\mathrm{tr}} \widetilde{F}$), we have $(E_i[\widetilde{F}])\langle\widetilde{a}\rangle \preceq_{\mathrm{tr}} \widetilde{F}\langle\widetilde{a}\rangle$; and by the substitutivity properties of $\preceq_{\mathrm{tr}}$, also $(E_i^n[\widetilde{F}])\langle\widetilde{a}\rangle \preceq_{\mathrm{tr}} \widetilde{F}\langle\widetilde{a}\rangle$. We conclude that s is also a trace of $\widetilde{F}\langle\widetilde{a}\rangle$. □

The proof of the Lemma 2.2.40 is very similar to the proof of Theorem 1.4.11. For more details, we refer the reader to [DHS19], specifically the proof of unique solution for weak bisimilarity in the setting of CCS.

Lemma 2.2.40 (Post-fixed points, $\preceq_{\mathrm{tr}}$)**.** *Let $\mathcal{E}$ be a guarded system of equations, and $\widetilde{\mathrm{K}}_{\mathcal{E}}$ its syntactic solution. Suppose $\widetilde{\mathrm{K}}_{\mathcal{E}}$ has no divergence. If $\widetilde{F}$ is a post-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{\mathrm{K}}_{\mathcal{E}}$.*

Proof. For simplicity, we only give the proof for a single equation E, rather than a system of equations. The generalisation to systems of equations is straightforward.

Consider an equation E, a process abstraction F, and suppose $F \preceq_{\mathrm{tr}} E[F]$. We fix a set of fresh names $\widetilde{a}$, and write P for $F\langle\widetilde{a}\rangle$. If $\widetilde{\alpha} = \alpha_1 \dots \alpha_n$ is a finite trace of P (i.e, $P \stackrel{\widetilde{\alpha}}{\Longrightarrow}$), we build a growing sequence of sequence transitions of $E^n\langle\widetilde{a}\rangle$ and such that $E^n[F]\langle\widetilde{a}\rangle \stackrel{\alpha_1 \dots \alpha_{i_k}}{\Longrightarrow} E_n[F] \stackrel{\alpha_{i_k+1},\dots,\alpha_n}{\Longrightarrow} P_n$.

1. If a sequence of transitions labeled by $\widetilde{\alpha}$ are all transitions of the context $E^n\langle\widetilde{a}\rangle$, we stop and we have:
   $E^n\langle\widetilde{a}\rangle \stackrel{\widetilde{\alpha}}{\Longrightarrow}$ and thus $K_E\langle\widetilde{a}\rangle \stackrel{\widetilde{\alpha}}{\Longrightarrow}$ and $\widetilde{\alpha}$ is a trace of $K_E\langle\widetilde{a}\rangle$.

2. Otherwise there is an infinite sequence of transitions from $K_E\langle\widetilde{a}\rangle$ with visible actions $\alpha_1, \dots, \alpha_{i_k}$ for some k; therefore $K_E\langle\widetilde{a}\rangle$ has a divergence.

We now explain the construction of the sequence. Assume (i) : $E^n\langle\widetilde{a}\rangle \stackrel{\alpha_1,\dots,\alpha_{i_k}}{\Longrightarrow} E_n$ and (ii) : $E_n[F] \stackrel{\alpha_{i_k+1},\dots,\alpha_n}{\Longrightarrow}$. We proceed as follows:

- By (i) it follows that $E^{n+1}[F]\langle\widetilde{a}\rangle \stackrel{\alpha_1,\dots,\alpha_{i_k}}{\Longrightarrow} E_n[E[F]]$

- By (ii) and congruence of $\preceq_{\mathrm{tr}}$, it follows that $\alpha_{i_k+1}, \dots, s_n$ is a trace of $E_n\langle\widetilde{a}\rangle[F] \preceq_{\mathrm{tr}} E_n[E[F]]$

- We take for the new sequence of transitions the concatenation of the previous one, and the part of $E_n[E[F]] \stackrel{\alpha_{i_k+1},\dots,\alpha_n}{\Longrightarrow}$ that is a transition of the context $E_n[E]$. Since E is weakly guarded, this is not an empty sequence.

By 2 his construction has to stop, otherwise there would be a divergence. We conclude by 1. □

C.3 Abstract Formulation (Alternative version)

C.3.1 Sets of Operations and unique solution Theorem

This Appendix is dedicated to an previous version of the abstract formulation in Section 1.3.1, initially stated in [DHS19]. Among the main differences with the results from Chapter 1, results are only formulated for functions with a single argument (monadic contexts), the formulation and results are simpler, but are also less powerful, as discussed in Section 1.3.1.

The results of this section, up to Theorem C.3.6, have been formalised in Coq theorem prover [Dur17].

Definition C.3.1 (Autonomy)**.** For state functions f, f' we say that there is an *autonomous μ-transition from f to f'*, written $f \xrightarrow{\mu} f'$, if for all states x it holds that $f(x) \xrightarrow{\mu} f'(x)$.

Likewise, given a set $\mathcal{F}$ of state functions and $f \in \mathcal{F}$, we say that a transition $f(x) \xrightarrow{\mu} y$ is *autonomous on $\mathcal{F}$* if, for some $f' \in \mathcal{F}$ we have $f \xrightarrow{\mu} f'$ and $y = f'(x)$. Moreover, we say that *function f is autonomous on $\mathcal{F}$* if all the transitions emanating from f (that is, all transitions of the form $f(x) \xrightarrow{\mu} y$, for some x, μ, y) are autonomous on $\mathcal{F}$.

When $\mathcal{F}$ is clear, we omit it, and we simply say that a function *is autonomous*.

Thus, f is autonomous on $\mathcal{F}$ if, for some indexing set I, there are μ_i and functions $f_i \in \mathcal{F}$ such that for all x it holds that: $f(x) \xrightarrow{\mu_i} f_i(x)$, for each i; the set of all transitions emanating from $f(x)$ is precisely $\cup_i \{f(x) \xrightarrow{\mu_i} f_i(x)\}$. Autonomous transitions correspond to expression transitions in CCS, and autonomous functions correspond to guarded contexts, which do not need the contribution of their process argument to perform the first transition.

We now formulate conditions under which, intuitively, a state function behaves like a CCS context. Functions satisfying these conditions are called *operations*. Such operators are defined relative to a behavioural equivalence or preorder; in this section we are interested in bisimilarity, hence the relation R in the definition below should be understood to be $\approx$.

Definition C.3.2 (Set of operations)**.** Consider an LTS $\mathcal{T}$, a binary relation R on $\mathcal{T}$, and a set $\mathcal{O}$ of functions from S to S. We say that $\mathcal{O}$ is a *set of R-operations on $\mathcal{T}$* if the following conditions hold:

1. $\mathcal{O}$ contains the identity function;
2. $\mathcal{O}$ is closed by composition (that is, $f \circ g \in \mathcal{O}$ whenever $f, g \in \mathcal{O}$);
3. composition preserves autonomy (i.e., if g is autonomous on $\mathcal{O}$, then so is $f \circ g$);
4. R is preserved, that is, all functions in $\mathcal{O}$ respect R

A 'symmetric variant' of clause 3 always holds: if f is autonomous, then so is $f \circ g$: indeed, transitions of $f(x)$ do not depend on x, hence transitions of $f(g(x))$ do not depend on either x or g. Clause 4 expresses congruence of the equivalence w.r.t. the set of contexts: here, the state functions in $\mathcal{O}$. The autonomous transitions of a set of R-operations yield an LTS whose states are the operations themselves. Such transitions are of the form $f \xrightarrow{\mu} g$.

In the remainder of this section, we assume R to be equal to bisimilarity, and call *set of operators* any set of $\approx$-operations, without specifying the relation. Where the underlying set $\mathcal{O}$ of operations is clear, we simply call a function belonging to $\mathcal{O}$ an *operation*.

We define weak transitions as follows: $f \Rightarrow g$ if there is a sequence of autonomous transitions $f \xrightarrow{\tau} f_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} f_n \xrightarrow{\tau} g$. $\stackrel{\mu}{\Rightarrow}$ is then simply $\Rightarrow\xrightarrow{\mu}\Rightarrow$.

We use state functions to express equations, such as $X = f(X)$. We thus look at conditions under which such an equation has a unique solution (again, the generalisation to a system of equations is easy, using n-ary functions).

Thinking of functions as equation expressions, to formulate our abstract theory about unique solution of equations, we have to define the divergences of finite and infinite unfoldings of state functions. The nth unfolding of f (for $n \geq 1$), f^n, is the function obtained by n applications of f. An operator is *well-behaved* if there is n with f^n autonomous (the well-behaved operators correspond, in CCS, to equations some finite unfolding of which yields a guarded expression). We also have to reason about the infinite unfolding of an equation $X = f(X)$. For this, given a set $\mathcal{O}$ of operations, we consider the infinite terms obtained by infinite compositions of operations in $\mathcal{O}$, that is, the set coinductively defined by the grammar:

$$F := f \circ F \qquad \text{for } f \in \mathcal{O}$$

(Note that this syntax only deals with infinite compositions. Since $\mathcal{O}$ is closed under finite compositions, we do not need to handle such compositions in the above definition.) In particular, we write f^∞ for the infinite term $f \circ f \circ f \circ \ldots$.

We define the autonomous transitions for such infinite terms using the following rules:

$$\frac{g \xrightarrow{\mu} g'}{g \circ F \xrightarrow{\mu} g' \circ F}\ g \text{ autonomous} \qquad\qquad \frac{(g \circ f) \circ F \xrightarrow{\mu} F'}{g \circ (f \circ F) \xrightarrow{\mu} F'}\ g \text{ not autonomous}$$

Intuitively a term is 'unfolded', with the second rule, until an autonomous function is uncovered, and then transitions are computed using the first rule (we disallow unnecessary unfoldings; these would complicate our abstract theorems, by adding duplicate transitions, since the transitions of $g \circ f$ duplicate those of g when g is autonomous). An infinite term has no transitions if none of its finite unfoldings ever yields an autonomous function. This situation does not arise for terms of the form f^∞ or $g \circ f^\infty$, where f is well-behaved, which are the terms we are interested in. Note that no infinite term belongs to a set of operators.

These rules are consistent for finite compositions of functions in $\mathcal{O}$, in the sense that they allow one to infer the correct transitions for finite compositions of functions.

The following lemmas show that infinite terms have the same transitions $\xrightarrow{\mu}$ and weak transitions $\stackrel{\mu}{\Rightarrow}$ as their counterpart finite unfoldings. To build a transition $f^\infty \xrightarrow{\mu} F$ from a transition of an unfolding $f^n \xrightarrow{\mu} g$, we need to reason up to (finite) unfoldings of f: thus we set $=_f$ to be the symmetric reflexive transitive closure of the relation that relates g and g' whenever $g = g' \circ f$.

Lemma C.3.3. *Let f be a well-behaved operation and g an operation in some set $\mathcal{O}$. Then $g \circ f^\infty \xrightarrow{\mu} F$ for some F if and only if there is n such that $g \circ f^n \xrightarrow{\mu} h$ for some h; in which case, there is $h' =_f h$ such that $F = h' \circ f^\infty$.*

Proof. ($\Rightarrow$) Assume $g \circ f^\infty \xrightarrow{\mu} F$. By a simple induction over the derivation of this transition, we get that there is n such that f^n is autonomous, $f^n \xrightarrow{\mu} h$, and $F = h \circ f^\infty$

($\Leftarrow$) Assume $g \circ f^n \xrightarrow{\mu} h$. Since f is well-behaved and composition respects autonomy, there is m such that $g \circ f^m$ is autonomous; we take that m to be the minimum among the exponents that make $g \circ f^m$ autonomous. If $n \leq m$, there is an autonomous transition $g \circ f^m \xrightarrow{\mu} h \circ (f^{m-n})$. If $m \leq n$, given that $g \circ f^m$ is autonomous, there is a transition $g \circ f^m \xrightarrow{\mu} h'$, where $h' \circ f^{n-m} = h$. Either way, $g \circ f^m \xrightarrow{\mu} h_0$, where $h_0 =_f h$.

Since m is the smallest exponent that makes $g \circ f^m$ autonomous, we can derive $(g \circ f^m) \circ f^\infty \xrightarrow{\mu} g_0 \circ f^\infty$. This concludes the proof.

□

We now adapt Lemma C.3.3 to weak transitions.

Lemma C.3.4. *Let f be a well-behaved operation and g an operation in some set $\mathcal{O}$. Then $g \circ f^\infty \stackrel{\mu}{\Longrightarrow} F$ for some F if and only if there is n such that $g \circ f^n \stackrel{\mu}{\Longrightarrow} h$ for some h; in which case, there is $h' =_f h$ such that $F = h' \circ f^\infty$.*

Proof. For each implication, proceed by induction over the length of $\stackrel{\mu}{\Longrightarrow}$, and use Lemma C.3.3.
□

Definition C.3.5 (Operations and divergences)**.** Let f, f', f_i be operations in a set $\mathcal{O}$ of operations, and consider the LTS induced by the autonomous transitions of operations in $\mathcal{O}$. A sequence of transitions $f_1 \xrightarrow{\mu_1} f_2 \xrightarrow{\mu_2} f_3 \ldots$ is a *divergence* if for some $n \geq 1$ we have $\mu_i = \tau$ whenever $i \geq n$. We also say that f_1 *diverges*. We apply these notations and terminology also to infinite terms,
as expected.

In the remainder of the section we fix a set $\mathcal{O}$ of operations and we only consider autonomous transitions on $\mathcal{O}$. We now state the "abstract version" of Theorem 1.1.7.

Theorem C.3.6 (Unique solution, abstract formulation)**.** *Let f be a well-behaved operation on $\mathcal{O}$ on some LTS $\mathcal{T}$. If f^∞ does not diverge, then the equation $X = f(X)$ either has no solution or has a unique solution for $\approx$.*

Proof. The proof is essentially the same as that of Theorem 1.1.7, replacing equations expressions with operators, from a fixed set of operators $\mathcal{O}$ (one still has to consider reducts from an operator), and replacing instantiation of equation expression transitions with instantiation of autonomous transitions. A guarded equation expression becomes an autonomous operator. □

The equation in the statement of the theorem might have no solution at all. For example, consider the LTS $(\mathbb{N}, \{a\}, \rightarrow)$ where for each n we have $n + 1 \xrightarrow{a} n$. The function f with $f(n) = n + 1$ is an operation of the set $\mathcal{O} = \{f^n\}_{n \in \mathbb{N}}$ (with $f^0 = \mathtt{Id}$, the identity function). The function f is autonomous because, for all n, the only transition of $f(n)$ is $f(n) \xrightarrow{a} n$

(this transition is autonomous because $f \xrightarrow{a} \mathtt{Id}$). A fixpoint of f would be an element x with $x \xrightarrow{a} x$, and there is no such x in the LTS.

Theorem C.3.6 can be refined along the lines of Theorem 1.1.9. For this, we have to relate the divergences of any f^n (for $n \geq 1$) to divergences of f^∞, in order to distinguish between innocuous and non-innocuous divergences.

Lemma C.3.7. *Consider an autonomous operation f on $\mathcal{O}$ and a divergence of f^n*

$$f^n \xrightarrow{\mu_1} f_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_i} f_i \xrightarrow{\tau} f_{i+1} \xrightarrow{\tau} \ldots$$

This yields a divergence of f^∞: $f^\infty \xrightarrow{\mu_1} g_1 \circ f^\infty \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_i} g_i \circ f^\infty \xrightarrow{\tau} g_{i+1} \circ f^\infty \xrightarrow{\tau} \ldots$ such that for all $i \geq 1$, g_i is an operator and $g_i =_f f_i$.

Proof. The proof is a simple induction on the sequence of transitions defining the divergence. At each step, we unfold f^∞ as many times as needed. □

Given a divergence Δ of f^n, we write Δ^∞ to indicate the divergence of f^∞ obtained from Δ as in Lemma C.3.7. We call a divergence of f^∞ *innocuous* when it can be described in this way, that is, as a divergence Δ^∞ obtained from a divergence Δ of f^n, for some n.

Theorem C.3.8 (Unique solution with innocuous divergences, abstract formulation)**.** *Let $f \in \mathcal{O}$ be a well-behaved operation. If all divergences of f^∞ are innocuous, then the equation $X = f(X)$ either has no solution or has a unique solution for $\approx$.*

Proof. Just as in CCS, where the proof of Theorem 1.1.7 has to be modified for Theorem 1.1.9, here the proof of Theorem C.3.6 is to be modified. The modification is essentially the same as in CCS, again substituting equation expressions for operators. □

C.3.2 Trace equivalence and innocuous divergences

The notion of trace is extended to operations like we do for weak transitions: we write $f \stackrel{s}{\Longrightarrow}$ if there is a sequence of autonomous transitions $f \stackrel{\mu_1}{\Longrightarrow} f_1 \stackrel{\mu_2}{\Longrightarrow} \cdots \stackrel{\mu_n}{\Longrightarrow} f_n$ $(s = \mu_1, \ldots, \mu_n)$, and likewise for infinite traces.

Lemma C.3.9. *Given operations $g, \widetilde{f}$ and a (finite) trace s, $g \circ \widetilde{f}^\infty \stackrel{s}{\Longrightarrow}$ if and only if there is n such that $g \circ \widetilde{f}^n \stackrel{s}{\Longrightarrow}$.*

Proof. We proceed by induction over the length of s, and apply Lemma C.3.4 in each case. □

All theorems obtained for $\approx$ can be adapted to $\approx_{tr}$, with similar proofs. As an example, Theorem 1.2.21 becomes:

Theorem C.3.10. *Let $\widetilde{f} \in \mathbf{C}$ be operations of arity n such that operations in $\mathbf{C}$ respect $\approx_{tr}$. If all divergences of $\widetilde{f}^\infty$ are innocuous, then the equation $\widetilde{X} = \widetilde{f}(\widetilde{X})$ has at most one solution for $\approx_{tr}$.*

Proof. For simplicity, we consider the proof only in the case of a single equation $X = f(X)$ (extension to systems of equations is as in proof of Theorem 1.2.16).

We proceed by showing that $x \subseteq_{tr} f(x)$ implies $x \subseteq_{tr} f^\infty$, and then that $f(x) \subseteq_{tr} x$ implies $f^\infty \subseteq_{tr} x$. This indeed gives that $x \approx_{tr} f(x)$ implies $x \approx_{tr} f^\infty$; hence the equation has at most one solution (f^∞ does not belong to the LTS, hence it is not a solution).

1. $f(x) \subseteq_{tr} x$ implies $f^\infty \subseteq_{tr} x$. For this part, the absence of divergence hypothesis is not needed.

   Let s be a trace, and assume $f^\infty \stackrel{s}{\Rightarrow}$. By Lemma C.3.9, there is n such that $f^n \stackrel{s}{\Rightarrow}$. Hence, $f^n(x) \stackrel{s}{\Rightarrow}$. Since $f(x) \subseteq_{tr} x$ and f respects $\subseteq_{tr}$, we have that $f^n(x) \subseteq_{tr} f^{n-1}(x) \subseteq_{tr} \ldots \subseteq_{tr} x$. Hence, $x \stackrel{s}{\Rightarrow}$, and $f^\infty \subseteq_{tr} x$.

2. $x \subseteq_{tr} f(x)$ implies $x \subseteq_{tr} f^\infty$. This part of the proof is very similar to the proofs of Theorems 1.1.9 and 1.2.21.

   Assume $x \subseteq_{tr} f(x)$, and $x \stackrel{s}{\Rightarrow}$. We want to show that there exists some n such that $f^n \stackrel{s}{\Rightarrow}$, then apply Lemma C.3.9; this would show $f^\infty \stackrel{s}{\Rightarrow}$. To that end, we build a strictly increasing sequence of (autonomous) transitions $f^{p_n} \stackrel{s_n}{\Longrightarrow} g_n$, such that $g_n(x) \stackrel{s'_n}{\Longrightarrow}$ and such that $s = s_n s'_n$ for all n (s_n and s'_n are traces whose concatenation is s). Strictly increasing means that the transition $f^{(n+1)\times p} \stackrel{s_{n+1}}{\Longrightarrow} g_{n+1}$ can be decomposed as $f^{(n+1)\times p} \stackrel{s_n}{\Longrightarrow} g_n \circ f^p \stackrel{s''_n}{\Longrightarrow} g_{n+1}$ for some s''_n ($s_{n+1} = s_n s''_n$). Here $(p_n)_{n\in\mathbb{N}}$ is strictly increasing as well. This construction will have to stop, otherwise we build a non-innocuous divergence.

   Construction of the sequence. Assume p_0 is such that f^{p_0} is autonomous; such a p_0 exists, since f is does not diverge, by Lemma 1.2.10.

   We initialise with the empty trace from $f^0 = \mathbf{id}$. Indeed, we have $\mathbf{id}(x) \Rightarrow \mathbf{id}(x) \stackrel{s}{\Rightarrow}$, and s is indeed the concatenation of itself with the empty trace.

   Then, at step n, suppose we have, for instance, $f^{p_n} \stackrel{s_n}{\Longrightarrow} g_n$, and $g_n(x) \stackrel{s'_n}{\Longrightarrow}$.

   - If s'_n is the empty trace, we stop. We have in this case $f^n \stackrel{s}{\Rightarrow}$, which concludes.
   - Otherwise, choose k such that $g_n \circ f^k$ is autonomous (exists by Lemma 1.2.10). Then p_{n+1} is $p_n + k$. We have $f^{p_{n+1}} \stackrel{s_n}{\Longrightarrow} g_n \circ f^k$ (by autonomy). The operations respect $\subseteq_{tr}$, and by hypothesis $x \subseteq_{tr} f(x)$, therefore $g_n(x) \subseteq_{tr} g_n \circ f^k(x)$. From there, $g_n \circ f^k(x) \stackrel{s'_n}{\Longrightarrow}$.

     s'_n is not the empty sequence, so there is $y_1, \ldots, y_n$ for $n \geq 1$ such that $g_n \circ f^k(x) \xrightarrow{\mu_1} y_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} y_n$, where $s'_n = \mu_1 \ldots \mu_k$. Take $\mu_1 \ldots \mu_i$ the longest prefix of s'_n such that there is operations $h_1, \ldots, h_i$ such that $g_n \circ f^k \xrightarrow{\mu_1} h_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_i} h_i$. It cannot be empty: indeed, $g_n \circ f^k$ is autonomous. We take g_{n+1} to be h_i, s'_{n+1} to be the trace corresponding to $\mu_{i+1} \ldots \mu_k$ (i.e., removing all τs) and s_{n+1} to be $s_n s''_n$, where s''_n is the trace corresponding to $\mu_1 \ldots \mu_i$ (again, removing all τs).

Then, we indeed have $f^{p_{n+1}} = f^{p_n} \circ f^k \xRightarrow{s_{n+1}} g_{n+1}$, and $g_{n+1}(x) \xRightarrow{s'_{n+1}}$, where $s = s_{n+1}s'_{n+1}$.

Suppose now that the construction given above never stops. To make the argument clearer, we reason up to $=_f$ (we identify all operations that are $=_f$). We know that $g_n \circ f^k \xRightarrow{s''_n} g_{n+1}$ for some k, therefore, by applying Lemma C.3.4 as many times as needed (just as in Lemma C.3.9), we get that $g_n \circ f^\infty \xRightarrow{s''_n} g_{n+1} \circ f^\infty$.

This gives an infinite sequence of transitions starting from f^∞: $f^\infty \xRightarrow{s''_0} g_1 \circ f^\infty \xRightarrow{s''_1} \dots$. We observe that in the latter sequence, every step involves at least one transition (the s''_i are supposed non-empty, otherwise we would have stopped). Moreover, the sequence $s''_1 s''_2 \dots$, when removing all τs, is a prefix of s. Therefore there is a finite number of visible actions occurring in this infinite sequence. Therefore f^∞ is divergent.

Furthermore, this divergence is not innocuous: at every step, we know that the transition $g_{n+1}(x) \xRightarrow{s''_{n+1}}$ is not autonomous, otherwise s''_{n+1} would be part of s_n. Hence, there cannot be such a divergence $f^m \xRightarrow{s''_0} \xRightarrow{s''_1} \dots$, as this would imply $g_{n+1}(x) \xRightarrow{s''_{n+1}}$ is autonomous for n such that $p_n \geq m$.

□

Appendix D

Calculi and equations for the proof of Full Abstraction

D.1 Proofs about the encoding in Iπ

We present here the calculation details of the proofs of the various results in Section 2.2.6.

D.1.1 Properties of the encoding

Lemma 2.2.10. *We have:*

1. $\boldsymbol{\nu} q\,(p \triangleright q \mid q \triangleright r) \succeq p \triangleright r$, *for all continuation names* p, r.
2. $\boldsymbol{\nu} y\,(x \triangleright y \mid y \triangleright z) \succeq x \triangleright z$, *for all trigger names* x, z.

Proof. We first show the two following laws:

$$\begin{aligned}\boldsymbol{\nu} q\,(p \triangleright q \mid q \triangleright r) &\sim p(x).\,\boldsymbol{\nu} q\,(\overline{q}(y).\,y \triangleright x \mid q \triangleright r)\\ &\succeq p(x).\,\boldsymbol{\nu} y\,(y \triangleright x \mid \overline{r}(z).\,z \triangleright y)\\ &\sim p(x).\,\overline{r}(z).\,\boldsymbol{\nu} y\,(z \triangleright y \mid y \triangleright x)\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{\nu} y\,(x \triangleright y \mid y \triangleright z) &\sim !x(p, x').\,\boldsymbol{\nu} y\,(\overline{y}(q, y').\,(y' \triangleright x' \mid q \triangleright p) \mid y(q, y').\,\overline{z}(r, z').\,(z' \triangleright y' \mid r \triangleright q))\\ &\succeq !x(p, x').\,\boldsymbol{\nu} q, y'\,.\,(y' \triangleright x' \mid q \triangleright p \mid \overline{z}(r, z').\,(z' \triangleright y' \mid r \triangleright q))\\ &\sim !x(p, x').\,\overline{z}(r, z').\,(\boldsymbol{\nu} q\,(r \triangleright q \mid q \triangleright p) \mid \boldsymbol{\nu} y'\,(z' \triangleright y' \mid y' \triangleright x'))\end{aligned}$$

We define a relation $\mathcal{R}$ as the relation that contains, for all continuation names p, r and for all trigger names x, z, the following pairs:

1. $\boldsymbol{\nu} q\,(p \triangleright q \mid q \triangleright r)$ and $p \triangleright r$
   $\boldsymbol{\nu} y\,(x \triangleright y \mid y \triangleright z)$ and $x \triangleright z$

2. $\overline{r}(z).\boldsymbol{\nu} y\,(z \triangleright y \mid y \triangleright x)$ and $\overline{r}(z).\, z \triangleright x$
   $\overline{z}(r, z').\,(\boldsymbol{\nu} q\,(r \triangleright q \mid q \triangleright p) \mid \boldsymbol{\nu} y'\,(z' \triangleright y' \mid y' \triangleright x'))$ and $\overline{z}(r, z').\,(r \triangleright p \mid z' \triangleright x')$
   (for all non-continuation names x or x', z')

We show that this is an expansion up to expansion and contexts, using the previous laws (each of those processes has only one possible action).

□

Lemma 2.2.12. *We have:*

1. $\boldsymbol{\nu} x\,(\mathcal{I}[\![M]\!]\langle p\rangle \mid x \triangleright y) \succeq \mathcal{I}[\![M\{y/x\}]\!]\langle p\rangle$;

2. $\boldsymbol{\nu} p\,(\mathcal{I}[\![M]\!]\langle p\rangle \mid p \triangleright q) \succeq \mathcal{I}[\![M]\!]\langle q\rangle$;

3. $\boldsymbol{\nu} y\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid x \triangleright y) \succeq \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle$.

Proof. 1. By induction over the structure of M. We use Lemma 2.2.10, and usual laws of expansion $\succeq$ [SW01].

- If $M = z$, $z \neq x$, then $\boldsymbol{\nu} x\,(\mathcal{I}[\![z]\!]\langle p\rangle \mid x \triangleright y) = \boldsymbol{\nu} x\,(\overline{p}(z').\, z' \triangleright z \mid x \triangleright y) \sim \overline{p}(z').\, z' \triangleright z = \mathcal{I}[\![z]\!]\langle p\rangle = \mathcal{I}[\![z\{y/x\}]\!]\langle p\rangle$ (because x does not appear free in $\overline{p}(z').\, z' \triangleright z$).
- If $M = x$, then :

$$\begin{aligned}
\boldsymbol{\nu} x\,(\mathcal{I}[\![x]\!]\langle p\rangle \mid x \triangleright y) &= \boldsymbol{\nu} x\,(\overline{p}(z).\, z \triangleright x \mid x \triangleright y) \\
&\sim \overline{p}(z).\, \boldsymbol{\nu} x\,(z \triangleright x \mid x \triangleright y) \\
&\succeq \overline{p}(z).\, z \triangleright y \text{ (by lemma 2.2.10)} \\
&= \mathcal{I}[\![y]\!]\langle p\rangle \\
&= \mathcal{I}[\![x\{y/x\}]\!]\langle p\rangle
\end{aligned}$$

- If $M = \lambda z.\, M'$:

$$\begin{aligned}
\boldsymbol{\nu} x\,(\mathcal{I}[\![\lambda z.\, M']\!]\langle p\rangle \mid x \triangleright y) &= \boldsymbol{\nu} x\,(\overline{p}(z').\, !z'(z, q).\, \mathcal{I}[\![M']\!]\langle q\rangle \mid x \triangleright y) \\
&\sim \overline{p}(z').\, !z'(z, q).\, \boldsymbol{\nu} x\,(\mathcal{I}[\![M']\!]\langle q\rangle \mid x \triangleright y) \\
&\succeq \overline{p}(z').\, !z'(z, q).\, \mathcal{I}[\![M'\{y/x\}]\!]\langle q\rangle \text{ (by induction hypothesis)} \\
&= \mathcal{I}[\![\lambda z.\, (M'\{y/x\})]\!]\langle p\rangle \\
&= \mathcal{I}[\![(\lambda z.\, M')\{y/x\}]\!]\langle p\rangle
\end{aligned}$$

- If $M_1\ M_2$:

$$\begin{aligned}
\boldsymbol{\nu} x\,(\mathcal{I}[\![M_1\ M_2]\!]\langle p\rangle \mid x \triangleright y) &= \boldsymbol{\nu} x\,(\boldsymbol{\nu} q\,(\mathcal{I}[\![M_1]\!]\langle q\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![M_2]\!]\langle r\rangle \mid r(w).\,\overline{u}(w', r').\\
&\qquad (w' \triangleright w \mid r' \triangleright p))) \mid x \triangleright y)\\
&\sim \boldsymbol{\nu} q\,(\boldsymbol{\nu} x\,(\mathcal{I}[\![M_1]\!]\langle q\rangle \mid x \triangleright y)\\
&\qquad \mid q(u).\,\boldsymbol{\nu} r\,(r(w).\,\overline{u}(w', r').\,(w' \triangleright w \mid r' \triangleright p)\\
&\qquad\quad \mid \boldsymbol{\nu} x\,(\mathcal{I}[\![M_2]\!]\langle r\rangle \mid x \triangleright y)))\\
&\succeq \boldsymbol{\nu} q\,(\mathcal{I}[\![M_1\{y/x\}]\!]\langle q\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![M_2\{y/x\}]\!]\langle r\rangle \mid r(w).\,\overline{u}(w', r').\\
&\qquad (w' \triangleright w \mid r' \triangleright p))) \text{ (by induction hypothesis)}\\
&= \mathcal{I}[\![(M_1\{y/x\})\ (M_2\{y/x\})]\!]\langle p\rangle\\
&= \mathcal{I}[\![(M_1\ M_2)\{y/x\}]\!]\langle p\rangle
\end{aligned}$$

2. By induction over the structure of M. We also use the previous case.

- If $M = x$:

$$\begin{aligned}
\boldsymbol{\nu} p\,(\mathcal{I}[\![x]\!]\langle p\rangle \mid p \triangleright q) &= \boldsymbol{\nu} p\,(\overline{p}(y).\,y \triangleright x \mid p \triangleright q)\\
&\succeq \boldsymbol{\nu} y\,(y \triangleright x \mid \overline{q}(z).\,z \triangleright y)\\
&\sim \overline{q}(z).\,\boldsymbol{\nu} y\,(z \triangleright y \mid y \triangleright x)\\
&\succeq \overline{q}(z).\,z \triangleright x \text{ (by Lemma 2.2.10)}\\
&= \mathcal{I}[\![x]\!]\langle q\rangle
\end{aligned}$$

- If $M = \lambda x.\,M'$:

$$\begin{aligned}
\boldsymbol{\nu} p\,(\mathcal{I}[\![\lambda x.\,M']\!]\langle p\rangle \mid p \triangleright q) &= \boldsymbol{\nu} p\,(\overline{p}(y).\,!y(x, r).\,\mathcal{I}[\![M']\!]\langle r\rangle \mid p \triangleright q)\\
&\succeq \boldsymbol{\nu} y\,(!y(x, r).\,\mathcal{I}[\![M']\!]\langle r\rangle \mid \overline{q}(z).\,z \triangleright y)\\
&\sim \overline{q}(z).\,!z(w, p).\,\boldsymbol{\nu} y\,(!y(x, r).\,\mathcal{I}[\![M']\!]\langle r\rangle \mid \overline{y}(x, r).\,(w \triangleright x \mid r \triangleright p))\\
&\succeq \overline{q}(z).\,!z(w, p).\,\boldsymbol{\nu} x, r\,(\mathcal{I}[\![M']\!]\langle r\rangle \mid w \triangleright x \mid r \triangleright p)\\
&\succeq \overline{q}(z).\,!z(w, p).\,\mathcal{I}[\![M'\{x/w\}]\!]\langle p\rangle\\
&\qquad \text{(by induction hypothesis and using case 1)}\\
&= \mathcal{I}[\![\lambda x.\,M']\!]\langle q\rangle
\end{aligned}$$

- If $M = M_1\ M_2$:

$$\begin{aligned}
\boldsymbol{\nu} p\,(\mathcal{I}[\![M_1\ M_2]\!]\langle p\rangle \mid p \triangleright s) &= \boldsymbol{\nu} p\,(\boldsymbol{\nu} q\,(\mathcal{I}[\![M]\!]\langle q\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{u}(w', r').\\
&\qquad (w' \triangleright w \mid r' \triangleright p))) \mid p \triangleright s)\\
&\sim \boldsymbol{\nu} q\,(\mathcal{I}[\![M]\!]\langle q\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{u}(w', r').\\
&\qquad (w' \triangleright w \mid \boldsymbol{\nu} p\,(r' \triangleright p \mid p \triangleright s))))\\
&\succeq \boldsymbol{\nu} q\,(\mathcal{I}[\![M]\!]\langle q\rangle \mid q(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{u}(w', r').\\
&\qquad (w' \triangleright w \mid r' \triangleright s))) \text{ (by Lemma 2.2.10)}\\
&= \mathcal{I}[\![M_1\ M_2]\!]\langle s\rangle
\end{aligned}$$

3. Law 3 can be derived from laws 1 and 2, by case analysis on V. Two cases:

   - If $V = \lambda x.\, M$, then $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle = !y(w,p).\, \mathcal{I}[\![M\{w/x\}]\!]\langle p\rangle$; then
   $$\begin{aligned}\boldsymbol{\nu} y\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid x \triangleright y) &\sim !x(z,q).\, \boldsymbol{\nu} w,p\,(\mathcal{I}[\![M\{w/x\}]\!]\langle p\rangle \mid p \triangleright q \mid w \triangleright z)\\ &\succeq !x(z,q).\, \mathcal{I}[\![M\{z/x\}]\!]\langle q\rangle \text{ (by cases 1 and 2)}\\ &= \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle\end{aligned}$$
   - If $V = z$: then $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle y\rangle = !y(w,q).\, \overline{z}(w',q').\, (w' \triangleright w \mid q' \triangleright q) = y \triangleright z$; we can then conclude by Lemma 2.2.10.

□

Lemma 2.2.13 (Validity of β_{v}-reduction)**.** *For any M, N in Λ, $M \longrightarrow N$ implies $\mathcal{I}[\![M]\!] \succeq \mathcal{I}[\![N]\!]$.*

Proof. One shows $\mathcal{I}[\![(\lambda x.\, M)V]\!] \succeq \mathcal{I}[\![M\{V/x\}]\!]$ exploiting algebraic properties of replication; then the result follows by the compositionality of the encoding and the congruence of $\succeq$.

We start by simplifying $\mathcal{I}[\![(\lambda x.\, M)V]\!]$:

$$\begin{aligned}\mathcal{I}[\![(\lambda x.\, M)V]\!]\langle p\rangle = \boldsymbol{\nu} q\,(&\overline{q}(y).\, !y(x,s).\, \mathcal{I}[\![M]\!]\langle s\rangle \mid q(y).\, \boldsymbol{\nu} r\,(\\ &\overline{r}(w).\, \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle\\ &\mid r(w).\, \overline{y}(w',p').\, (w' \triangleright w \mid p' \triangleright p)))\\ \succeq \boldsymbol{\nu} y\,(&!y(x,p').\, \mathcal{I}[\![M]\!]\langle p'\rangle \mid \boldsymbol{\nu} w\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(x,p').\, (x \triangleright w \mid p' \triangleright p)))\\ \succeq \boldsymbol{\nu} x,p'\,(&\mathcal{I}[\![M]\!]\langle p'\rangle \mid p' \triangleright p \mid \boldsymbol{\nu} w\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid x \triangleright w))\\ \succeq \boldsymbol{\nu} x\,(&\mathcal{I}[\![M]\!]\langle p\rangle \mid \boldsymbol{\nu} w\,(!w(z,s).\, \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z,s\rangle \mid x \triangleright w)) \text{ (by Lemma 2.2.12)}\\ \sim \boldsymbol{\nu} x\,(&\mathcal{I}[\![M]\!]\langle p\rangle \mid \boldsymbol{\nu} w\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid x \triangleright w))\\ \succeq \boldsymbol{\nu} x\,(&\mathcal{I}[\![M]\!]\langle p\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \text{ (by Lemma 2.2.12)}\end{aligned}$$

We now prove, by induction over M, that

$$\boldsymbol{\nu} x\,(\mathcal{I}[\![M]\!]\langle p\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \succeq \mathcal{I}[\![M\{V/x\}]\!]\langle p\rangle$$

:

- If $M = z$, $z \neq x$:
$$\begin{aligned}\boldsymbol{\nu} x\,(\mathcal{I}[\![z]\!]\langle p\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) &= \boldsymbol{\nu} x\,(\overline{p}(u).\, u \triangleright z \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle)\\ &\sim \overline{p}(u).\, u \triangleright z \text{ (because } \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle = !x(y,q)\dots)\\ &= \mathcal{I}[\![z\{V/x\}]\!]\langle p\rangle\end{aligned}$$
- If $M = x$:
$$\begin{aligned}\boldsymbol{\nu} x\,(\mathcal{I}[\![x]\!]\langle p\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) &= \boldsymbol{\nu} x\,(\overline{p}(z).\, z \triangleright x \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle)\\ &\sim \overline{p}(z).\, \boldsymbol{\nu} x\,(z \triangleright x \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle)\\ &\succeq \overline{p}(z).\, \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \text{ (by Lemma 2.2.12)}\\ &= \mathcal{I}[\![V]\!]\langle p\rangle\\ &= \mathcal{I}[\![x\{V/x\}]\!]\langle p\rangle\end{aligned}$$

- If $M = \lambda w.\, M'$:

$$\begin{aligned}
\boldsymbol{\nu} x\, (\mathcal{I}[\![\lambda w.\, M']\!]\langle p\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) &= \boldsymbol{\nu} x\, (\overline{p}(z).\, !z(w,r).\, \mathcal{I}[\![M']\!]\langle r\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \\
&\sim \overline{p}(z).\, !z(w,r).\, \boldsymbol{\nu} x\, (\mathcal{I}[\![M']\!]\langle r\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \\
&\succeq \overline{p}(z).\, !z(w,r).\, \mathcal{I}[\![M'\{V/x\}]\!]\langle r\rangle \text{ (by induction hypothesis)} \\
&= \mathcal{I}[\![\lambda w.\, (M'\{V/x\})]\!]\langle p\rangle \\
&= \mathcal{I}[\![(\lambda w.\, M')\{V/x\}]\!]\langle p\rangle
\end{aligned}$$

- If $M = M_1 M_2$. First recall that $\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle$ is always replicated, so:

$$\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle \sim \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle$$

  We deduce:

$$\begin{aligned}
\boldsymbol{\nu} x\, (\mathcal{I}[\![M_1 M_2]\!]\langle p\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) &= \boldsymbol{\nu} x\, (\boldsymbol{\nu} q\, (\mathcal{I}[\![M_1]\!]\langle q\rangle \mid q(u).\, \boldsymbol{\nu} r\, (\mathcal{I}[\![M_2]\!]\langle r\rangle \\
&\quad \mid r(w).\, \overline{u}(w', r').\, (w' \triangleright w \mid r' \triangleright p))) \\
&\quad \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \\
&\sim \boldsymbol{\nu} q\, (\boldsymbol{\nu} x\, (\mathcal{I}[\![M_1]\!]\langle q\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \\
&\quad \mid q(u).\, \boldsymbol{\nu} r\, (\boldsymbol{\nu} x\, (\mathcal{I}[\![M_2]\!]\langle r\rangle \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle x\rangle) \\
&\quad \mid r(w).\, \overline{u}(w', r').\, (w' \triangleright w \mid r' \triangleright p))) \\
&\succeq \boldsymbol{\nu} q\, (\mathcal{I}[\![M_1\{V/x\}]\!]\langle q\rangle \mid q(u).\, \boldsymbol{\nu} r\, (\mathcal{I}[\![M_2\{V/x\}]\!]\langle r\rangle \\
&\quad \mid r(w).\, \overline{u}(w', r').\, (w' \triangleright w \mid r' \triangleright p))) \\
&\quad \text{(by induction hypothesis, applied on } M_1 \text{ and } M_2) \\
&= \mathcal{I}[\![M_1\{V/x\} M_2\{V/x\}]\!]\langle p\rangle \\
&= \mathcal{I}[\![(M_1 M_2)\{V/x\}]\!]\langle p\rangle
\end{aligned}$$

We therefore have that $\mathcal{I}[\![(\lambda x.\, M)V]\!]\langle p\rangle \succeq \mathcal{I}[\![M\{V/x\}]\!]\langle p\rangle$; by the compositionality of the encoding and the congruence of $\succeq$, it follows that for all evaluation context C_{e}

$$\mathcal{I}[\![C_{\mathrm{e}}[(\lambda x.\, M)V]]\!] \succeq \mathcal{I}[\![C_{\mathrm{e}}[M\{V/x\}]]\!]$$

which concludes. □

D.1.2 Soudness

Lemma 2.2.14. *We have:*

$$\mathcal{O}[\![C_{\mathrm{e}}[xV]]\!]\langle p\rangle \succeq \overline{x}(z,q).\, (\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\, \mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle).$$

Proof. By induction on the evaluation context C_{e}.

- **Base case:** C_{e} is the empty evaluation context $[\cdot]$. We have to show that

$$\mathcal{I}[\![xV]\!]\langle p\rangle \succeq \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![y]\!]\langle p\rangle)$$

We first remark that $q(y).\,\mathcal{I}[\![y]\!]\langle p\rangle = q \triangleright p$.

$$\begin{aligned}
\mathcal{I}[\![xV]\!]\langle p\rangle &= \boldsymbol{\nu} p\,(\overline{q}(y).\,y \triangleright x \mid q(y).\,\boldsymbol{\nu} r\,(\overline{r}(w).\,\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid r(w).\,\overline{y}(w',p').\,(w' \triangleright w \mid p' \triangleright p)))\\
&\succeq \boldsymbol{\nu} y\,(y \triangleright x \mid \boldsymbol{\nu} w\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(w',p').\,(w' \triangleright w \mid p' \triangleright p)))\\
&\succeq \boldsymbol{\nu} w', p'\,(\overline{x}(z,q).\,(z \triangleright w' \mid q \triangleright p') \mid \boldsymbol{\nu} w\,(!w(z,s).\,\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid w' \triangleright w \mid p' \triangleright p))\\
&\succeq \overline{x}(z,q).\,\boldsymbol{\nu} w\,(z \triangleright w \mid q \triangleright p \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle w\rangle) \text{ (by Lemma 2.2.10)}\\
&\succeq \overline{x}(z,q).\,(q \triangleright p \mid \mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle)) \text{ (by Lemma 2.2.12)}\\
&\sim \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![y]\!]\langle p\rangle)
\end{aligned}$$

- Case $C_{\mathrm{e}} = C'_{\mathrm{e}}t$

$$\begin{aligned}
\mathcal{I}[\![C_{\mathrm{e}}[xV]]\!]\langle p\rangle &= \mathcal{I}[\![C'_{\mathrm{e}}[xV]t]\!]\langle p\rangle\\
&= \boldsymbol{\nu} s\,(\mathcal{I}[\![C'_{\mathrm{e}}[xV]]\!]\langle s\rangle \mid s(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![t]\!]\langle r\rangle \mid r(w).\,\overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p)))\\
&\succeq \boldsymbol{\nu} s\,(\overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![C'_{\mathrm{e}}[y]]\!]\langle s\rangle) \mid\\
&\quad s(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![t]\!]\langle r\rangle \mid r(w).\,\overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p)))\\
&\quad \text{(by induction hypothesis)}\\
&\sim \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\boldsymbol{\nu} s\,(\\
&\quad (\mathcal{I}[\![C'_{\mathrm{e}}[y]]\!]\langle s\rangle \mid s(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![t]\!]\langle r\rangle \mid r(w).\,\overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p)))))\\
&= \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![C'_{\mathrm{e}}[y]t]\!]\langle p\rangle)\\
&= \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle)
\end{aligned}$$

- Case $C_{\mathrm{e}} = V'C'_{\mathrm{e}}$

$$\begin{aligned}
\mathcal{I}[\![C_{\mathrm{e}}[xV]]\!]\langle p\rangle &= \mathcal{I}[\![V'C'_{\mathrm{e}}[xV]]\!]\langle p\rangle\\
&= \boldsymbol{\nu} s\,(\mathcal{I}[\![V']\!]\langle s\rangle \mid s(u).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![C'_{\mathrm{e}}[xV]]\!]\langle r\rangle \mid r(w).\,\overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p)))\\
&\succeq \boldsymbol{\nu} s\,(\overline{s}(u).\,\mathcal{I}_{\mathrm{V}}[\![V']\!]\langle u\rangle \mid s(u).\\
&\quad \boldsymbol{\nu} r\,((\overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![C'_{\mathrm{e}}[y]]\!]\langle r\rangle) \mid r(w).\,\overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p)))\\
&\quad \text{(by induction hypothesis)}\\
&\succeq \boldsymbol{\nu} u\,(\mathcal{I}_{\mathrm{V}}[\![V']\!]\langle u\rangle \mid \boldsymbol{\nu} r\,(\overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![C'_{\mathrm{e}}[y]]\!]\langle r\rangle)\\
&\quad \mid r(w).\,\overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p)))\\
&\sim \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![C'_{\mathrm{e}}[y]]\!]\langle r\rangle \mid r(w).\,\boldsymbol{\nu} u\,(\mathcal{I}_{\mathrm{V}}[\![V']\!]\langle u\rangle\\
&\quad \mid \overline{u}(w',r').\,(w' \triangleright w \mid r' \triangleright p))))) \text{ (because } \mathcal{I}_{\mathrm{V}}[\![V']\!]\langle u\rangle = !u(v,s)\ldots)\\
&= \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![V'C'_{\mathrm{e}}[y]]\!]\langle p\rangle)\\
&= \overline{x}(z,q).\,(\mathcal{I}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{I}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle)
\end{aligned}$$

□

Lemma 2.2.15. *For all $M \in \Lambda$, $\mathcal{I}[\![M]\!] \succeq \mathcal{O}[\![M]\!]$.*

Proof. By induction over M.

- Case $M = x$: by definition, $\mathcal{O}[\![M]\!]\langle p\rangle = \mathcal{I}[\![M]\!]\langle p\rangle$
- Case $M = \lambda x.\, N$: assuming $\mathcal{O}[\![N]\!]\langle q\rangle \approx \mathcal{I}[\![N]\!]\langle q\rangle$, we have, by definition

$$\begin{aligned}\mathcal{O}[\![M]\!]\langle p\rangle &= \overline{p}(y).\, !y(x,q).\, \mathcal{O}[\![N]\!]\langle q\rangle \\ &\sim \overline{p}(y).\, !y(x,q).\, \mathcal{I}[\![N]\!]\langle q\rangle \\ &= \mathcal{I}[\![M]\!]\langle p\rangle\end{aligned}$$

- Case $M = xV$:

$$\begin{aligned}\mathcal{I}[\![xV]\!]\langle p\rangle &= \boldsymbol{\nu} q\, (\mathcal{I}[\![x]\!]\langle q\rangle \mid q(y).\, \boldsymbol{\nu} r\, (\mathcal{I}[\![V]\!]\langle r\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \\ &\succeq \boldsymbol{\nu} q\, (\mathcal{I}[\![x]\!]\langle q\rangle \mid q(y).\, \boldsymbol{\nu} r\, (\mathcal{O}[\![V]\!]\langle r\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \quad \text{(IH)} \\ &\succeq \boldsymbol{\nu}(y, w)\, (y \triangleright x \mid \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p)) \\ &\succeq \boldsymbol{\nu}(w, w')\, (\overline{x}(z, q).\, (z \triangleright w' \mid q \triangleright p') \mid \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid w' \triangleright w \mid p' \triangleright p) \\ &\succeq \overline{x}(z, q).\, \boldsymbol{\nu}(w, w')\, (\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid z \triangleright w' \mid w' \triangleright w \mid q \triangleright p' \mid p' \triangleright p) \\ &\succeq \overline{x}(z, q).\, \boldsymbol{\nu} w\, (\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid z \triangleright w \mid q \triangleright p) \\ &\succeq \overline{x}(z, q).\, (\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q \triangleright p)\end{aligned}$$

- Case $M = (\lambda x.\, N)V$:

$$\begin{aligned}\mathcal{I}[\![(\lambda x.\, N)V]\!]\langle p\rangle &= \boldsymbol{\nu} q\, (\overline{q}(y).\, !y(x, p').\, \mathcal{I}[\![N]\!]\langle p'\rangle \mid q(y).\, \boldsymbol{\nu} r\, (\mathcal{I}[\![V]\!]\langle r\rangle \mid \\ &\qquad r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \\ &\succeq \boldsymbol{\nu} q\, (\overline{q}(y).\, !y(x, p').\, \mathcal{I}[\![N]\!]\langle p'\rangle \mid q(y). \\ &\qquad \boldsymbol{\nu} r\, (\overline{r}(w).\, \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \quad \text{(IH)} \\ &\succeq \boldsymbol{\nu}(y, p', w)\, (!y(x, p').\, \mathcal{I}[\![N]\!]\langle p'\rangle \mid \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(x, p').\, (x \triangleright w \mid p' \triangleright p)) \\ &\succeq \boldsymbol{\nu}(y, p', w)\, (!y(x, p').\, \mathcal{O}[\![N]\!]\langle p'\rangle \mid \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(x, p').\, (x \triangleright w \mid p' \triangleright p)) \quad \text{(IH)} \\ &= \mathcal{O}[\![M]\!]\langle p\rangle\end{aligned}$$

- Case $M = VN$:

$$\begin{aligned}\mathcal{I}[\![VN]\!]\langle p\rangle &= \boldsymbol{\nu} q\, (\mathcal{I}[\![V]\!]\langle q\rangle \mid q(y).\, \boldsymbol{\nu} r\, (\mathcal{I}[\![N]\!]\langle r\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \\ &\succeq \boldsymbol{\nu} q\, (\overline{q}(y).\, \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid q(y).\, \boldsymbol{\nu} r\, (\mathcal{O}[\![N]\!]\langle r\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \quad \text{(IH)} \\ &\succeq \boldsymbol{\nu} y\, (\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle y\rangle \mid \boldsymbol{\nu} r\, (\mathcal{O}[\![N]\!]\langle r\rangle \mid r(w).\, \overline{y}(w', p').\, (w' \triangleright w \mid p' \triangleright p))) \\ &= \mathcal{O}[\![VN]\!]\langle p\rangle\end{aligned}$$

- Case $M = NV$:

$$\begin{aligned}
\mathcal{I}[\![NV]\!]\langle p\rangle &= \boldsymbol{\nu} q\,(\mathcal{I}[\![N]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} r\,(\mathcal{I}[\![V]\!]\langle r\rangle \mid r(w).\,\overline{y}(w', p').\,(w' \triangleright w \mid p' \triangleright p))) \\
&\succeq \boldsymbol{\nu} q\,(\mathcal{O}[\![N]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} r\,(\overline{r}(w).\,\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle r\rangle \mid r(w).\,\overline{y}(w', p').\,(w' \triangleright w \mid p' \triangleright p))) \quad \text{(IH)} \\
&\succeq \boldsymbol{\nu} q\,(\mathcal{O}[\![N]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} w\,(\mathcal{O}_{\mathrm{V}}[\![V]\!]\langle w\rangle \mid \overline{y}(w', p').\,(w' \triangleright w \mid p' \triangleright p))) \\
&= \mathcal{O}[\![NV]\!]\langle p\rangle
\end{aligned}$$

- Case $M = M'N$: the encodings are the same.

□

Lemma 2.2.16 (Operational correspondence)**.** *For any $M \in \Lambda$ and fresh p, process $\mathcal{O}[\![M]\!]\langle p\rangle$ has exactly one immediate transition, and exactly one of the following clauses holds:*

1. $\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\overline{p}(y)} P$ *and M is a value, with* $P = \mathcal{O}_{\mathrm{V}}[\![M]\!]\langle y\rangle$*;*
2. $\mathcal{O}[\![M]\!]\langle p\rangle \xrightarrow{\overline{x}(z,q)} P$ *and* $M = C_{\mathrm{e}}[xV]$ *and*
$$P \succeq \mathcal{O}_{\mathrm{V}}[\![V]\!]\langle z\rangle \mid q(y).\,\mathcal{O}[\![C_{\mathrm{e}}[y]]\!]\langle p\rangle;$$
3. $\mathcal{O}_{\mathrm{V}}[\![M]\!]\langle p\rangle \xrightarrow{\tau} P$ *and there is N with $M \longrightarrow N$ and $P \succeq \mathcal{O}[\![N]\!]\langle p\rangle$.*

Proof. By induction on M. We use Lemmas 2.2.13, 2.2.10, and 2.2.12. □

D.1.3 Completeness: systems of equations

We provide the full description of the systems of equations $\mathcal{E}_{\mathcal{R}}$ (Figure D.1) and $\mathcal{E}'_{\mathcal{R}}$ (Figure D.2). There, $\widetilde{y}$ is assumed to be the ordering of $\mathsf{fv}(M, N)$. In $\mathcal{E}'_{\mathcal{R}}$ we write $\widetilde{y}'$ or $\widetilde{y}''$ for the free variables of the terms indexing the corresponding equation variable.

For both systems, we give all equations needed to handle $\leftrightarrows_\eta$. As explained in Section 2.2.7, the systems to handle $\leftrightarrows$ are obtained by omitting some equations (precisely, the last two equations in Figures D.1 and D.2).

$$
\begin{array}{ll}
M \Uparrow \text{ and } N \Uparrow\text{:} & X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\Omega]\!] \\
M \Downarrow x \text{ and } N \Downarrow x : & X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![x]\!] \\
M \Downarrow \lambda x.\, M' \text{ and } N \Downarrow \lambda x.\, N' : & X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\lambda x.\, X_{M',N'}]\!] \\
M \Downarrow C_{\mathrm{e}}[xV] \text{ and } N \Downarrow C'_{\mathrm{e}}[xV'] : & X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![(\lambda z.\, X_{C_{\mathrm{e}}[z],C'_{\mathrm{e}}[z]})(xX_{V,V'})]\!] \\
M \Downarrow x,\ N \Downarrow \lambda z.\, N',\ N' \Downarrow C_{\mathrm{e}}[xV] : & X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\lambda z.\, \big((\lambda w.\, X_{w,C_{\mathrm{e}}[w]})(xX_{z,V})\big)]\!] \\
M \Downarrow \lambda z.\, M',\ M' \Downarrow C_{\mathrm{e}}[xV],\ N \Downarrow x : & X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\lambda z.\, \big((\lambda w.\, X_{C_{\mathrm{e}}[w],w})(xX_{V,z})\big)]\!]
\end{array}
$$

Figure D.1: System $\mathcal{E}_{\mathcal{R}}$ of equations (the last two equations are included only when considering $\leftrightarrows_\eta$)

$$
\begin{array}{ll}
M \Uparrow \text{ and } N \Uparrow\text{:} & \\
\qquad X_{M,N} = (\widetilde{y}, p)\ \mathbf{0} & \\
M \Downarrow C_{\mathrm{e}}[xV] \text{ and } N \Downarrow C'_{\mathrm{e}}[xV'] : & \\
\qquad X_{M,N} = (\widetilde{y}, p)\ \overline{x}(z,q).\, (X^{\mathcal{V}}_{V,V'}\langle z, \widetilde{y}'\rangle \mid q(w).\, X_{C_{\mathrm{e}}[w],C'_{\mathrm{e}}[w]}\langle \widetilde{y}'', p\rangle) & \\
M \Downarrow V \text{ and } N \Downarrow V' : & \\
\qquad X_{M,N} = (\widetilde{y}, p)\ \overline{p}(y).\, X^{\mathcal{V}}_{v,v'}\langle z, \widetilde{y}'\rangle & \\
V = x \text{ and } V' = x : & \\
\qquad X^{\mathcal{V}}_{x,x} = (z, x)\ z \triangleright x & \\
V = \lambda x.\, M \text{ and } V' = \lambda x.\, N : & \\
\qquad X^{\mathcal{V}}_{\lambda x.M,\lambda x.N} = (z, \widetilde{y})\ !z(x,q).\, X_{M,N}\langle \widetilde{y}', q\rangle & \\
V = x,\ V' = \lambda z.\, N,\ N \Downarrow C_{\mathrm{e}}[xV] : & \\
\qquad X^{\mathcal{V}}_{x,\lambda z.N} = (y_0, \widetilde{y})\ !y_0(z,q).\, \overline{x}(z',q').\, (X^{\mathcal{V}}_{z,V}\langle z', \widetilde{y}'\rangle \mid q'(w).\, X_{w,C_{\mathrm{e}}[w]}\langle \widetilde{y}'', q\rangle) & \\
V = \lambda z.\, M,\ M \Downarrow C_{\mathrm{e}}[xV],\ V' = x : & \\
\qquad X^{\mathcal{V}}_{\lambda z.M,x} = (y_0, \widetilde{y})\ !y_0(z,q).\, \overline{x}(z',q').\, (X^{\mathcal{V}}_{V,z}\langle z', \widetilde{y}'\rangle \mid q'(w).\, X_{C_{\mathrm{e}}[w],w}\langle \widetilde{y}'', q\rangle) &
\end{array}
$$

Figure D.2: System $\mathcal{E}'_{\mathcal{R}}$ of equations (the last two equations are needed only when considering $\leftrightarrows_\eta$)