École Polytechnique
Thèse de Doctorat
Spécialité Informatique

A linear approach to the proof-theory of least and greatest fixed points

Présentée et soutenue publiquement par

David Baelde

le 9 décembre 2008

devant le jury composé de

Patrick	Baillot	Rapporteur
Gilles	Dowek	
Olivier	Laurent	
Paul-André	Melliès	
Dale	Miller	Directeur de thèse
Gopalan	Nadathur	
Christine	Paulin-Mohring	Rapporteur
Frank	Pfenning	Rapporteur

To my parents and siblings.

Acknowledgments

The past three years as a PhD student have been interesting, exciting and enjoyable. For that, I thank my adviser Dale Miller very much. He provided me with wise scientific advice, and has always been available for interesting discussions. It has been a chance and a pleasure to work with Dale. I learned a lot from him and I hope that that our collaboration does not end with this thesis. Very importantly, Dale brought me to a lively research environment among his team and collaborators, who I thank for their insightful comments and communicative enthusiasm: Kaustuv Chaudhuri, Andrew Gacek, Gopalan Nadathur, Frank Pfenning, Brigitte Pientka, Alexis Saurin, and especially Alwen Tiu on the footsteps of whom I walked.

I am honored that Patrick Baillot, Christine Paulin-Mohring and Frank Pfenning have accepted to review my thesis, and I thank them for their careful reading. I am delighted that Paul-André Melliès and Gilles Dowek accepted to be part of my jury, they have a deep and broad scientific vision and I look forward to hearing their comments. I would like to thank especially Olivier Laurent for taking part to my jury, and for the helpful and insightful discussions that we had. Finally, I would like to thank François Pottier and Luigi Santocanale, who unfortunately could not be part of my jury, for their interest in my work; I particularly enjoyed the discussions with Luigi, which opened new perspectives for me.

Big thanks to the great people from LIX: Antoine, Romain and Sylvain from the balloons office, Olivier, Vivek and Alexis from the Tarski-and-Church office, Nicolas, Zach and Alexandre, Vincent, Florian, David, Catuscia, Frank, Miki, Christoph, Luc. I am very thankful to the administrative and technical staff who make the LIX go round: in particular, Catherine and Isabelle, Mathieu and James. Big thanks as well to the great people from Lyon: Samuel, Julien, Stéphane and Martin, the two Florent, Sylvain, Marc, Etienne, Jade and our wonderful godfather Daniel Hirschkoff.

I warmly thank my old friends without which (these past few) years would have been less fun: Fabien, Noémi, Juliette, and especially Nicolas. Much more than thanks to Estelle, whose support (*encourager, soutenir, supporter*) makes anything sweet.

But there is more than a handful of people to thank, and, although I could not name them all, I am truly thankful to all the people who guided, pushed and taught me for years. This includes my family, friends and teachers — M$^{\text{me}}$ Boussel, M. Choquet, if you ever read this…

Contents

Introduction

The scientific discourse, and the mathematical one above all, is characterized by the production of *proofs*, which are supposed to be flawless reasoning. The task of early logicians was to study the logical discourse in order to understand when it is valid. Beyond that question, modern logicians develop and study the mathematical structure of various logics and notions of proof. Reasoning about a finite universe of objects or possibilities is essentially combinatorial: there might be a large amount of cases to consider, and several ways to enumerate them all, but it is always possible to check them one by one. This process can be well delimited and understood. But the logical discourse is a much more powerful tool than that, with which man can tame infinity, reasoning at once about an inaccessible amount of possibilities. The task of the logician, concerning these aspects, is more difficult.

Mathematicians widely use sets, generally infinite ones. It is only at the beginning of the XX$^{\text{th}}$ century that they started to study the structure of infinity, with the foundation of set theory by Cantor. This made them realize that their intuition of sets and infinity was often misleading, and raised the question of what are the valid axioms in that domain. Eventually, axiomatizations of set theory were proposed, and were proved (relatively) consistent. But there still does not seem to be one canonical list of axioms that can be used to carry all set-theoretic proofs. Indeed, Gödel and Cohen showed that principles such as the axiom of choice or the continuum hypothesis were *independent*: set theory is consistent with either the axiom or its negation. There is no canonical structure to the general infinity that sets incarnate.

There are less problematic infinities. Let us consider one of the oldest mathematical concepts: 0, 1, 2... natural numbers. They have certainly been at the center of philosophical disputes, even until the XVII$^{\text{th}}$ century where mathematicians such as Fermat and Pascal disagreed on the valid methods for reasoning about natural numbers. Pascal essentially used induction as we know it, but Fermat criticized it, preferring the *infinite descent* technique that Euclid used in his proof of the irrationality of $\sqrt{2}$. However, we now know that both techniques are perfectly correct, and correspond well to the common (faithful) intuition of natural numbers. Both derive from the same fact: the canonical ordering of natural numbers is *well-founded*, *i.e.,* it admits no infinite descending chain. This common property comes from the underlying *structure* of natural numbers. The set of natural numbers is indeed built from the empty set by iteratively adding 0 and the successors of already present numbers. The obtained collection is

a *least fixed point*, and the ordering of natural numbers corresponds to their order of construction.

Least fixed point constructions, more commonly called *inductive definitions*, are widely used in computer science: lists, formulas, but also relations such as typing or evaluation judgments are defined inductively. For each of these constructions, the generalized induction principle applies: "If a property holds for any object under the assumption that it holds for its predecessors in the sense of the iterated construction, then it holds for any object". Once this general concept has been identified, it only takes one step to consider the dual notion associated to *greatest* fixed points: one obtains coinductive definitions, and the associated deductive principle of coinduction. They are increasingly used in computer science, for example, to define infinite data types such as streams, or behavioral equivalences such as the bisimulation of π-calculus.

In this thesis, we consider a formal logical treatment of these two important concepts: least and greatest fixed points. More precisely, we shall take a *proof-theoretical* approach. Proof theory is the study of proofs as genuine mathematical objects. We shall work within the particular framework of *sequent calculus*, a formalism which offers an algebraic view of logic that is very principled and modular. In sequent calculus, the fundamental property of cut-elimination requires a tight structure on logics, a precise balance between the principles introducing (proving) and eliminating (reasoning about) a logical concept. Least and greatest fixed points can be satisfyingly treated within sequent calculus, because their structure is well characterized by the above mentioned deductive principles: inhabitants of least fixed points are obtained by finite iteration, and one reasons about them by induction; symmetrically, inhabitants of greatest fixed points are built by coinduction and analyzed by finite iteration.

More precisely, our approach is based on *linear logic*, a refinement of both classical and intuitionistic logics. Linear logic rejects *a priori* the ability to use the same hypothesis any number of times. The usual non-linear treatment of hypothesis, involving copies and erasures, is recovered by means of the exponential connectives. Linear logic, without exponentials, is thus a very finite system. By combining it with least and greatest fixed points, we obtain a simple framework that is well-suited to the proof-theoretical study of fixed points. We shall also see that it is a surprisingly powerful logic. In other words, fixed points are a sufficient source of infinity in themselves, and do not require exponentials for many applications that are usually conceived as intuitionistic. Linear logic is not only a simple framework, but also *the logic behind logic*. It is a powerful looking glass for studying common aspects of logics. Linear logic has notably allowed the discovery of focusing, an important property of the structure of proofs. This observation renewed the vision of logics and brought several advances in various domains. One of the contributions of this thesis is the extension of focusing to fixed points. It is first developed in the linear framework, then adapted to the more conventional intuitionistic setting.

Outline of the thesis

The first two chapters introduce the main concepts of the thesis:

- In Chapter 1 we introduce some basic notions of proof-theory that shall be useful in the following, notably sequent calculus, linear logic and focusing.

- Chapter 2 provides an introduction to the central concept of fixed points. We describe there the origins of our treatment of fixed points, its motivations and main problems, and briefly discuss other approaches.

The heart of the thesis is the development and study of the logic μMALL, which is carried in the next two chapters. Some of the contents of these two chapters has been published in [BM07].

- We introduce the logic μMALL in Chapter 3. The symmetry of this system makes it a good framework for studying proofs by induction and coinduction. We study the basic proof theory of μMALL, notably proving that it enjoys cut-elimination. We make some key observations about admissible structural rules, and the high expressiveness of our system.

- Chapter 4 contains our main development, that is the extension of focusing to fixed points. We design a focused system for μMALL, and provide a modular proof of its completeness which highlights the key mechanisms involved. We discuss why this system is satisfying, both from a theoretical and a practical point of view, but observe that an alternative focusing system is also possible. We show that our results extend to exponentials. Finally, we design a fragment of intuitionistic logic where the focused treatment of μMALL applies directly, and discuss the design of a satisfying focusing system for the full intuitionistic logic.

The last three chapters are relatively independent, building on the previous developments and validating them.

- In Chapter 5, we study how μMALL can be used to reason on finite automata inclusions, a natural class of complex problems. We show that fixed points allow for a natural correspondence, from which we obtain a novel characterization of automata inclusion as *multi-simulation*, and the completeness of μMALL for inclusions. We then apply our observations directly to the logic, by designing and studying the fragment of *regular* formulas. We obtain an internal completeness result which lays the foundations of new automated theorem proving techniques.

- Chapter 6 deals with generic quantification, a logical notion introduced by Miller and Tiu to reason richly about specifications involving variable bindings, *e.g.,* evaluation and typing in programming languages, provability in logics, etc. We come back to the original design of generic quantification and show that its poor interaction between fixed points and generic quantification causes a lack of expressiveness. We propose another design which treats generic quantification as a defined (non-logical) connective, and show that it restores the expected expressiveness without affecting the intended semantics. This work has been published in [Bae08a, Bae08b].

- The thesis ends with Chapter 7, in which we briefly describe how previous developments are applied in two implementations: the logic programming language Bedwyr, and the semi-automated theorem prover Taci which uses our focused proof-search strategy.

Chapter 1

Preliminaries

The field of *proof theory* studies *proofs*, or *derivations*, as genuine mathematical objects. There can be several notions of derivations for the same logic, as long as they prove all theorems and only theorems. In this chapter we introduce sequent calculus, a particular style of presentation for proof systems, and related notions. We shall in particular present the sequent calculus for linear logic and its focusing. Our goal here is not to give a formal introduction to sequent calculus, but rather to recall useful notions for this thesis and highlight some unusual aspects.

It is worth pointing out that sequent calculus is not the only syntax for proofs. It is a rather elegant one that comes with its own tools and corpus of knowledge, on top of which we shall work. For a number of questions, syntax does not matter. But we also address some truly syntactic questions in this thesis, among which focusing. Concerning these aspects, changing for an other formalism would have probably brought a different viewpoint and caused different problems. It is not in the scope of this thesis to investigate in that direction. Questions of syntax might seem shallow and frustrating, but they are essential to the effectiveness of proof theory, in particular for a computer scientist.

1.1 Syntax

Although we do not work within Church's type theory, we follow the same approach to syntax. We do not intend to deal with low-level aspects of concrete syntax. Instead, we leverage simply typed λ-calculus as a representational framework. Our languages shall be given by a set of constructors as typed constants, and formulas are considered up to $\beta\eta$-conversion; we shall in fact generally work with $\beta\eta$-long forms.

Like Church, we shall use the type o for *formulas* (or *truth values*), but we do not use the single sort ι for *terms* (or *individuals*). Instead, we consider any simple type in which o does not occur as a valid *term type*. We denote such types by γ. For example, formulas of first-order logic are obtained by taking the following constants:

$$\begin{array}{rcl} \top, \bot & : & o \\ \wedge, \vee, \supset & : & o \to o \to o \end{array}$$

$$\begin{aligned} \forall_\gamma, \exists_\gamma &: (\gamma \to o) \to o \\ =_\gamma &: \gamma \to \gamma \to o \end{aligned}$$

Writing the type annotations of the first-order formula constructors is cumbersome and of little interest, hence we shall omit them when they are irrelevant or can be determined from the context.

Using this *higher-order abstract syntax* [MN87, PE88] is convenient as it leverages types to distinguish terms from formulas, and λ-abstraction to deal with the notions of renaming (α-conversion) and substitution (β-reduction). Of course, we shall still write formulas using the usual concrete syntax, *e.g.,* $\forall x.\exists y.\ x = y \wedge x = x$. The point here is to describe what it denotes, *e.g.,* $\forall(\lambda x.\ \exists(\lambda y.\ (\wedge)\ ((=)\ x\ y)\ ((=)\ x\ x)))$.

We do not deny the importance of names, renaming and substitution. They are critical in a correct implementation of a logic, and play an important role in its efficiency — they shall indeed be discussed in Chapter 7, which is dedicated to our implementations. For the rest of the thesis, it is more convenient to be able to abstract these problems away and still remain formal.

The issues related to variable bindings are now sufficiently well understood by the community to be omitted most of the time in the semi-formal discourse of logicians. At this level it has indeed become common to accept the higher-order abstract syntax viewpoint. This contrasts a lot with the currently active research done to support good formalizations. We also address that aspect in Chapter 6, where we contribute to the field of reasoning directly on specifications using higher-order abstract syntax.

1.2 Sequent calculus

In 1935, Gentzen [Gen69] introduced *sequent calculus* with the proof system LK for classical logic. The great symmetry of his presentation allows for elegant results, and his methodology still lasts and proves useful beyond its initial scope, both in terms of the logic under consideration and of the properties being studied.

The sequent calculus LK is presented in Figure 1.1. Its inference rules derive *sequents* which are of the form $x_1, \ldots, x_k; H_1, \ldots, H_n \vdash C_1, \ldots, C_m$ and should be understood as $\forall x_1 \ldots \forall x_k.\ (H_1 \wedge \ldots \wedge H_n) \supset (C_1 \vee \ldots \vee C_m)$. The leftmost zone, separated by the semicolon, is called *signature* and denoted by Σ; it is left untouched in most rules and is then omitted for brevity. This reading of sequents with an explicit signature binding universal variables is not Gentzen's original point of view, but appeared only later; it fits naturally with our approach to syntax.

The rules of LK are organized in three groups. The *structural group* roughly expresses that the collections of formulas on both sides of the turnstile ($\vdash$) are sets. The rules *WL* and *WR* are called *weakening* rules; *CL* and *CR* are *contractions*; *XL* and *XR* are *exchanges*. The *logical group* is the core of reasoning. The applicability of its rules only depends on the outermost connective of one formula in the concluding sequent, which creates a tight connection between the notions of connective and logical rule. This shallow treatment is made possible by the sequent structure, which is used for opening up some trivial superficial structure of the formulas. For example, the left conjunction rule moves the formula-level conjunction at the sequent level which allows

Identity group

$$\frac{}{\Sigma; P \vdash P}\ \mathit{axiom} \qquad \frac{\Sigma; \Gamma \vdash \Delta, P \quad \Sigma; P, \Gamma \vdash \Delta}{\Sigma; \Gamma \vdash \Delta}\ \mathit{cut}$$

Logical group

$$\frac{}{\Sigma; \Gamma, \bot \vdash \Delta}\ \bot \qquad \frac{}{\Sigma; \Gamma \vdash \top, \Delta}\ \top$$

$$\frac{\Sigma; \Gamma, P_i \vdash \Delta}{\Sigma; \Gamma, P_0 \wedge P_1 \vdash \Delta}\ \wedge L_i \qquad \frac{\Sigma; \Gamma \vdash P, \Delta \quad \Gamma \vdash Q, \Delta}{\Sigma; \Gamma \vdash P \wedge Q, \Delta}\ \wedge R$$

$$\frac{\Sigma; \Gamma, P \vdash \Delta \quad \Sigma; \Gamma, Q \vdash \Delta}{\Sigma; \Gamma, P \vee Q \vdash \Delta}\ \vee L \qquad \frac{\Sigma; \Gamma \vdash P_i, \Delta}{\Sigma; \Gamma \vdash P_0 \vee P_1, \Delta}\ \vee R_i$$

$$\frac{\Sigma; \Gamma \vdash P, \Delta \quad \Sigma; \Gamma, Q \vdash \Delta}{\Sigma; \Gamma, P \supset Q \vdash \Delta}\ \supset L \qquad \frac{\Sigma; \Gamma, P \vdash Q, \Delta}{\Sigma; \Gamma \vdash P \supset Q, \Delta}\ \supset R$$

$$\frac{\Sigma \vdash t : \gamma \quad \Sigma; \Gamma, Pt \vdash \Delta}{\Sigma; \Gamma, \forall_\gamma x.\ Px \vdash \Delta}\ \forall L \qquad \frac{\Sigma, x : \gamma\ ;\ \Gamma \vdash Px, \Delta}{\Sigma; \Gamma \vdash \forall_\gamma x.\ Px, \Delta}\ \forall R$$

$$\frac{\Sigma, x : \gamma\ ;\ \Gamma, Px \vdash \Delta}{\Sigma; \Gamma, \exists_\gamma x.\ Px \vdash \Delta}\ \exists L \qquad \frac{\Sigma \vdash t : \gamma \quad \Sigma; \Gamma \vdash Pt, \Delta}{\Sigma; \Gamma \vdash \exists_\gamma x.\ Px, \Delta}\ \exists R$$

Structural group

$$\frac{\Sigma; \Gamma \vdash \Delta}{\Sigma; \Gamma, P \vdash \Delta}\ WL \qquad \frac{\Sigma; \Gamma \vdash \Delta}{\Sigma; \Gamma \vdash P, \Delta}\ WR$$

$$\frac{\Sigma; \Gamma, P, P \vdash \Delta}{\Sigma; \Gamma, P \vdash \Delta}\ CL \qquad \frac{\Sigma; \Gamma \vdash P, P, \Delta}{\Sigma; \Gamma \vdash P, \Delta}\ CR$$

$$\frac{\Sigma; \Gamma, Q, P \vdash \Delta}{\Sigma; \Gamma, P, Q \vdash \Delta}\ XL \qquad \frac{\Sigma; \Gamma \vdash Q, P, \Delta}{\Sigma; \Gamma \vdash P, Q, \Delta}\ XR$$

Term well-formedness

$$\frac{(x : \gamma) \in \Sigma}{\Sigma \vdash x : \gamma} \qquad \frac{\Sigma \vdash M : \gamma' \to \gamma \quad \Sigma \vdash N : \gamma'}{\Sigma \vdash MN : \gamma} \qquad \frac{\Sigma, x : \gamma' \vdash Mx : \gamma}{\Sigma \vdash (\lambda x.\ Mx) : \gamma' \to \gamma}$$

Figure 1.1: The LK sequent calculus for first-order classical logic

subsequent applications of rules on the topmost connectives of the two conjuncts. The *identity group* contains the only rules that require a notion of equality on formulas: axiom and cut. To check a proof starting with one of these rules, one has to check that two formulas are the same. The novelty of sequent calculus is the cut, dual of the axiom. It allows indirect reasoning, and makes it relatively easy to obtain the equivalence between LK and other presentations of classical logic.

The treatment of first-order quantifiers deserves some attention. The existential rules ($\exists R$ and $\forall L$) allow the substitution of the existentially bound variable by any well-formed term, often called *witness*. Well-formed terms given by the judgment $\Sigma \vdash t : \gamma$, are simply-typed λ-terms of type γ over the signature Σ. We shall omit the well-formedness judgments in the rest of this thesis. The universal rules ($\forall R$ and $\exists L$) use an *eigenvariable* for representing the newly introduced variable. Gentzen coined the term of *eigenvariable*, which for him it denoted a *new* variable, *i.e.,* one that is unused in the concluding sequent. Choosing a new variable ensures that there is no assumption made on it, so that the deduction rule can indeed be read as "for any variable …". Here we do not need such an assumption by relying on a more abstract viewpoint, following the approach to syntax adopted for formulas. We shall not detail how to represent sequents in simply-typed λ-calculus, by introducing a new type and constructors. This is an easy task which allows us to see the signature as a list of binders, with α-equivalence built-in. The rules $\forall R$ and $\exists L$ are thus a mere *mobility of binders* from formula to sequent, which rules out naming conflicts. Considering for example that $x; px \vdash \forall x.\, qx$ is identical to $x; px \vdash \forall y.\, qy$, the universal rule should behave equally on them, which rules out the premise $x, x; px \vdash qx$ that is not the same as $x, y; px \vdash qy$.

Several logics can be given a sequent calculus by adapting the structural rules of LK. It is the case of intuitionistic and relevant logic, but also linear logic to some extent. It is striking that LJ, a sound and complete sequent calculus for intuitionistic logic can be obtained only by ignoring the right contraction rule to LK, hence limiting the right hand-side zone to at most one formula. This observation foreshadows the linear decomposition of classical and intuitionistic negations which will explain the phenomenon.

Proposition 1.1 (Eliminability of non-atomic axioms)**.** *We call* atoms *the predicate constants, and shall extend the terminology to predicate variables when there are any. The axiom rule can be restricted to the atomic case,* i.e., *the axiom between two atoms.*

This important observation is proved by repeatedly expanding all non-atomic applications of the axiom, for example:

$$\frac{}{P \wedge Q \vdash P \wedge Q} \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{}{P \vdash P}}{P, Q \vdash P} \quad \dfrac{\dfrac{}{Q \vdash Q}}{P, Q \vdash Q}}{P, Q \vdash P \wedge Q}}{P \wedge Q \vdash P \wedge Q}$$

Non-atomic axioms will be redundant in all sequent calculi under consideration. In particular we shall consider logics without propositional constants, in which the axiom rule will thus be unnecessary. In that situation the question of the identity/equality of formulas becomes pointless and only their local behaviors matter.

Proposition 1.2 (Cut-elimination)**.** *The cut rule is* admissible*: any derivation of a given sequent can be transformed into one which does not make use of the cut rule.*

Cut-elimination is Gentzen's main result on LK. To prove it, he designed a system of elementary proof reductions which reduces all instances of cut and established its termination. The essential cases rely on the duality between corresponding left and right rules, for example:

$$\dfrac{\dfrac{\dfrac{\Pi}{\Gamma, P, Q \vdash \Delta}}{\Gamma, P \wedge Q \vdash \Delta} \quad \dfrac{\dfrac{\Pi_P}{\Gamma \vdash P, \Delta} \quad \dfrac{\Pi_Q}{\Gamma \vdash Q, \Delta}}{\Gamma \vdash P \wedge Q, \Delta}}{\Gamma \vdash \Delta} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Pi}{\Gamma, P, Q \vdash \Delta} \quad \dfrac{\dfrac{\Pi_Q}{\Gamma \vdash Q, \Delta}}{\Gamma, P \vdash Q, \Delta}}{\Gamma, P \vdash \Delta} \quad \dfrac{\Pi_P}{\Gamma \vdash P, \Delta}}{\Gamma \vdash \Delta}$$

Cut-elimination has been studied under various angles (*e.g.,* confluence, Curry-Howard isomorphism, complexity) and in various systems (*e.g.,* classical, intuitionistic, linear, polarized). But we do not need to elaborate on that topic as we mostly study cut-free derivations in our work. However, we do use cut-elimination to validate our interpretation of the logic, for example, in Chapter 5.

The consequences of cut-elimination are of interest. It implies the *consistency* of the logic. Let us assume the opposite and consider the smallest cut-free derivation with a conclusion of the form $\vdash (\bot)^n$, *i.e.,* a sequent with no left hand-side and n occurrences of $\bot$ on the right hand-side. Its first rule can only be a structural rule since no logical rule applies to $\bot$ on the right, hence its immediate subderivation also has a concluding sequent of the form $\vdash (\bot)^m$ which contradicts the minimality of the initial derivation. A similar argument can in fact yield a more general property.

Proposition 1.3 (Subformula property)**.** *All formulas occurring in a cut-free derivation are subformulas of formulas occurring in the conclusion.*

In a propositional setting, the notion of subformula is that of subtree. In first-order logic, we have to accept any instance Pt as a subformula of $\exists x.Px$ and $\forall x.Px$. In other words, the first-order notion of subformula corresponds to the propositional notion for the formulas obtained by erasing terms. A similar adjustment would be silly for second-order quantification. In second-order logic, the subformula property is instead restricted to sequents that are not above a second-order instantiation rule. Cut-elimination remains an important property, from which consistency follows immediately.

We have quickly presented LK and some general concepts about sequent calculus. In the following, we shall omit several details when writing sequent calculus rules; what we mean should be clear from the above presentation. Notably, we shall often omit the signature Σ. We also use a more high-level viewpoint on sequents that allows treating the cumbersome structural rules implicitly. Instead of considering the left and right hand-sides of the sequent as lists with explicit structural rules for manipulating it, we allow ourselves to treat them as sets or multisets. It is in fact a common thing to do, but its implications are sometimes forgotten. It does not affect provability, but causes some proofs to collapse because it blurs the distinction between identical formulas.

For example there are essentially two ways of proving $p \wedge p \supset p$ which correspond to the two projections, but they are identified when omitting the structural rules. It is not a problem for us since we do not work at that level of detail, which is a research topic in itself. But these aspects are important and should not be forgotten. Thus, we insist that this viewpoint on sequents is only a notational convenience and shall not be regarded as a definition of the essential nature of sequents. Although this abuse hides some information, we claim that the work presented here is still valid in a setting with some strict tracking of occurrences. Indeed, some of it has been implemented in that way.

1.3 Linear logic

Linear logic [Gir87] is a refinement of both classical and intuitionistic logics, providing an elegant unified framework for their study. It was invented by Girard from observations on the coherent semantics of λ-calculus. But for our purpose we can restrict to a purely syntactic presentation of linear logic. A sequent calculus for full linear logic is given in Figure 1.2; we progressively introduce it below.

Definition 1.4 (Formulas of LL)**.** The formulas of linear logic are given by the following syntax:

$$\begin{array}{rcl} P & ::= & P \otimes P \mid P \oplus P \mid \mathbf{1} \mid \mathbf{0} \mid !P \mid \exists x.Px \mid p\vec{t} \\ & \mid & P \parr P \mid P \,\&\, P \mid \bot \mid \top \mid ?P \mid \forall x.Px \mid (p\vec{t})^{\perp} \end{array}$$

We deliberately do not precise any types, especially for the quantifiers that can equally be first or second-order ones in this section.

Linear logic can be obtained by removing the arbitrary contraction and weakening rules of LK. Consider the two following rules:

$$\frac{\Gamma \vdash P, \Delta \quad \Gamma' \vdash Q, \Delta'}{\Gamma, \Gamma' \vdash P \wedge Q, \Delta, \Delta'} \qquad \frac{\Gamma \vdash P, \Delta \quad \Gamma \vdash Q, \Delta}{\Gamma \vdash P \wedge Q, \Delta}$$

In LK, the structural rules make it easy to derive one from the other. It does not hold anymore in a calculus without contraction and weakening. In such a setting, since a rule really corresponds to a connective, there should in fact be two conjunctions. The first one is called $\otimes$, the second one $\&$. Similarly, two disjunctions appear:

$$\frac{\Gamma \vdash P, Q, \Delta}{\Gamma \vdash P \parr Q, \Delta} \qquad \frac{\Gamma \vdash P_i, \Delta}{\Gamma \vdash P_0 \oplus P_1, \Delta}$$

Adapting the cut reductions to these new connectives shows that $\parr$ and $\otimes$ (resp. $\oplus$ and $\&$) are duals. The connectives $\parr$ and $\otimes$ are called *multiplicative*, as well as their respective units $\bot$ and $\mathbf{1}$. The connectives $\oplus$ and $\&$, together with their units $\mathbf{0}$ and $\top$ are called *additive* connectives. Together they form the multiplicative and additive fragment of linear logic, called MALL.

To obtain full linear logic, Girard added the unary connectives why-not (?) and of-course (!). Their role is to get back some non-linear behavior in an explicit way. The rules for $?P$ are *contraction*, *weakening* and *dereliction*:

$$\frac{\Gamma \vdash \Delta, ?P, ?P}{\Gamma \vdash \Delta, ?P} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, ?P} \qquad \frac{\Gamma \vdash \Delta, P}{\Gamma \vdash \Delta, ?P}$$

The only rule for $!P$ is the *promotion*, where $?\Delta$ denotes a multiset of formulas of the form $?P$, and similarly for $!\Gamma$:

$$\frac{!\Gamma \vdash ?\Delta, P}{!\Gamma \vdash ?\Delta, !P}$$

These two new connectives are called the *exponentials* because they turn additives in multiplicatives: $?(P \oplus Q) \equiv ?P \parr ?Q$ and dually.

At this point we roughly have all the ingredients of linear logic. We have not shown any left rule, but as is already visible in LK, the left rule of a connective is none but the right rule of its dual. This can be internalized in a simpler presentation using *mono-sided sequents*, since $\Gamma \vdash \Delta$ can be read as $\vdash \Gamma^\perp, \Delta$ where $\bullet^\perp$ denotes dualization, or negation.

Definition 1.5 (Negation)**.** The negation is the involution on formulas satisfying:

$$\begin{array}{rclcrcl}
(P \parr Q)^\perp & \equiv & P^\perp \otimes Q^\perp & \qquad & (P \,\&\, Q)^\perp & \equiv & P^\perp \oplus Q^\perp \\
(?P)^\perp & \equiv & !P^\perp & & (\forall x.\, Px)^\perp & \equiv & \exists x.\, (Px)^\perp \\
\bot^\perp & \equiv & \mathbf{1} & & \top^\perp & \equiv & \mathbf{0}
\end{array}$$

This definition allows pushing down negation through a formula, leaving it only on predicate constants and variables. If one needs to interpret negation on predicates, a bijection should be assumed between two disjoint sets covering all predicates so that negation is read as moving from one side to the other along that bijection.

Negation is not a *logical connective* but a *defined* one: it has no logical rule but is only defined in terms of other connectives. Similarly, the linear implication $P \multimap Q$ is defined as $P^\perp \parr Q$.

The sequent calculus for linear logic is given in Figure 1.2. It uses a simplified handling of exponentials that eliminates some irrelevant information. For example, with the original rules, $?P$ can be contracted ten times before that its copies get weakened. This is avoided here by organizing the sequent in two zones, written $\vdash \Theta; \Gamma$ and read as $\vdash ?\Theta, \Gamma$. Such sequents are called *dyadic*. The new non-linear zone Θ, treated as a set, allows a simpler treatment of formulas on which structural rules are available, by merging the contraction and dereliction rules into the action of moving one copy of a non-linear formula into the linear zone. This approach was developed by Andreoli [And92] in his Σ_2 system. When restricting to MALL, the non-linear zone can be ignored from the sequent calculus of Figure 1.2.

Linear logic can be seen as a *logic of resources* rather than truth, as formulas cannot *a priori* be contracted. From that point of view, the exponential connectives express re-usability or durability of a resource. Although it is indeed sometimes used as such [Mil93, HM94, Hod94, CP02, Bae05], it is not the main interest of linear logic. Instead, it is most often used as a *the logic behind logic*, a looking glass for studying our usual intuitionistic and classical logics.

Identity group

$$\frac{}{\vdash \Theta; P, P^{\perp}} \qquad \frac{\vdash \Theta; \Gamma, P \vdash \Theta; P^{\perp}, \Gamma'}{\vdash \Theta; \Gamma, \Gamma'}$$

Logical group

$$\frac{\vdash \Theta; \Gamma, P, Q}{\vdash \Theta; \Gamma, P \parr Q} \qquad \frac{\vdash \Theta; \Gamma, P \vdash \Theta; \Gamma', P'}{\vdash \Theta; \Gamma, P \otimes P', \Gamma'}$$

$$\frac{\vdash \Theta; \Gamma}{\vdash \Theta; \Gamma, \perp} \qquad \frac{}{\vdash \Theta; \mathbf{1}}$$

$$\frac{\vdash \Theta; \Gamma, P \quad \vdash \Theta; \Gamma, Q}{\vdash \Theta; \Gamma, P \mathbin{\&} Q} \qquad \frac{\vdash \Theta; \Gamma, P_i}{\vdash \Theta; \Gamma, P_0 \oplus P_1}$$

$$\frac{}{\vdash \Theta; \Gamma, \top} \qquad \text{no rule for } \mathbf{0}$$

$$\frac{\Sigma, x \vdash \Theta; \Gamma, Px}{\Sigma \vdash \Theta; \Gamma, \forall x.\, Px} \qquad \frac{\Sigma \vdash \Theta; \Gamma, Pt}{\Sigma \vdash \Theta; \Gamma, \exists x.\, Px}$$

$$\frac{\vdash \Theta, P; \Gamma}{\vdash \Theta; ?P, \Gamma} \qquad \frac{\vdash \Theta; P, \Gamma}{\vdash \Theta, P; \Gamma} \qquad \frac{\vdash \Theta; P}{\vdash \Theta; !P}$$

Figure 1.2: One-sided dyadic sequent calculus for LL

Definition 1.6 (Translation of intuitionistic logic [Gir87])**.** Intuitionistic connectives can be considered as defined in linear logic, as follows:

$$\begin{array}{rclcrcl} [P \wedge Q] & \equiv & [P] \,\&\, [Q] & & [P \vee Q] & \equiv & !{[P]} \oplus !{[Q]} \\ [\top] & \equiv & \top & & [\bot] & \equiv & \mathbf{0} \\ [P \supset Q] & \equiv & (!{[P]}) \multimap [Q] & & [p\ t] & \equiv & p\ t \\ [\forall x.\ Px] & \equiv & \forall x.\ [Px] & & [\exists x.\ Px] & \equiv & \exists x.\ [Px] \end{array}$$

This translation is proved sound by translating intuitionistic proofs in natural deduction to linear proofs using cuts. Completeness is obtained by translating cut-free linear proofs to intuitionistic sequent calculus. We shall see that different translations can be used, depending on one's interests: semantic soundness, simulation of cut-elimination, or simply soundness and completeness. Translations of classical logics also exist, starting with [Gir87], revolving around an interpretation of implication as $(!P) \multimap (?Q)$.

1.4 Focusing

Focusing [And92] is one of the major observations about linear logic. As is well-known, some rules are *invertible*: applying them never breaks provability. But Andreoli observed a symmetric phenomenon about the other rules, which are the real choices in proof-search: they can always be chained in an hereditary way. For example on $P \oplus (Q \oplus R)$, it does not break completeness to require that if one chooses the right disjunct, he has to immediately choose between Q and R. As an other example, consider a sequent containing both $A \otimes (B \otimes C)$ and $D \otimes E$. A naive backward proof-search might first consider all splittings of $A \otimes (B \otimes C)$; in one case it has to treat $B \otimes C$ and $D \otimes E$ in the same sequent. From this point the two tensors can be split in different ways and orders, resulting in four configurations, each being reached in two different ways. That state explosion is avoided by the focused strategy since it forces the splitting of $B \otimes C$ after that of $A \otimes (B \otimes C)$.

Andreoli used these observations in the design of his focused calculus. In this system, proof-search alternates between two phases: the *asynchronous* phase corresponds to invertible steps and the *synchronous* phase to chained choices. It starts with the asynchronous phase, in which invertible rules are applied eagerly in any order. Eventually, the linear part of the sequent contains only atoms and synchronous formulas. At this point a choice has to be made: the *focus* has to be set on one of these formulas. Once a focus is set, the system enters the synchronous phase in which rules are applied on the formula under focus, until it becomes asynchronous. Proof-search then falls back to the asynchronous phase.

Definition 1.7 (Synchronous, asynchronous)**.** The connectives $\parr, \&, \bot, \top, ?$ and $\forall$ (resp. $\otimes, \oplus, \mathbf{1}, \mathbf{0}, !$ and $\exists$) are classified as *asynchronous* (resp. synchronous). Atoms shall be given an arbitrary synchrony, consistent with negation: if a is synchronous, $a^\perp$ must be asynchronous and conversely. The synchrony of a rule is that of the associated connective; the synchrony of a formula is that of its toplevel connective or predicate. For simplicity, we shall simply write synchronous atoms as a and asynchronous ones as $a^\perp$.

The focused sequent calculus is given in Figure 1.3. It uses annotations on Σ_2 sequents (see Section 1.3, Figure 1.2) in order to constrain proof-search. In the synchronous phase, the formula under focus is distinguished in the linear zone: $\vdash \Theta; \Gamma \Downarrow P$. In the asynchronous phase, the linear zone is split in two parts: $\vdash \Theta; \Gamma \Uparrow \Delta$ where Γ already contains only atomic and synchronous formulas while Δ is still to be processed.

Asynchronous phase

$$\frac{\vdash \Theta; \Gamma \Uparrow P, Q, \Delta}{\vdash \Theta; \Gamma \Uparrow P \parr Q, \Delta} \qquad \frac{\vdash \Theta; \Gamma \Uparrow P, \Delta \quad \vdash \Theta; \Gamma \Uparrow Q, \Delta}{\vdash \Theta; \Gamma \Uparrow P \,\&\, Q, \Delta}$$

$$\frac{\vdash \Theta; \Gamma \Uparrow \Delta}{\vdash \Theta; \Gamma \Uparrow \bot, \Delta} \qquad \frac{}{\vdash \Theta; \Gamma \Uparrow \top, \Delta}$$

$$\frac{\vdash \Theta; \Gamma \Uparrow Px, \Delta}{\vdash \Theta; \Gamma \Uparrow \forall x.Px, \Delta} \qquad \frac{\vdash \Theta, P; \Gamma \Uparrow \Delta}{\vdash \Theta; \Gamma \Uparrow ?P, \Delta}$$

Synchronous phase

$$\frac{\vdash \Theta; \Gamma \Downarrow P \quad \vdash \Theta; \Gamma' \Downarrow Q}{\vdash \Theta; \Gamma, \Gamma' \Downarrow P \otimes Q} \qquad \frac{\vdash \Theta; \Gamma \Downarrow P_i}{\vdash \Theta; \Gamma \Downarrow P_0 \oplus P_1}$$

$$\frac{}{\vdash \Theta; \Downarrow \mathbf{1}}$$

$$\frac{\vdash \Theta; \Gamma \Downarrow Pt}{\vdash \Theta; \Gamma \Downarrow \exists x.Px} \qquad \frac{\vdash \Theta; \Uparrow P}{\vdash \Theta; \Downarrow !P}$$

$$\frac{}{\vdash \Theta, a^\perp; \Downarrow a} \qquad \frac{}{\vdash \Theta; a^\perp \Downarrow a}$$

Switching (where P is synchronous, Q asynchronous)

$$\frac{\vdash \Theta; \Gamma, P \Uparrow \Delta}{\vdash \Theta; \Gamma \Uparrow P, \Delta} \qquad \frac{\vdash \Theta; \Gamma \Downarrow P}{\vdash \Theta; \Gamma, P \Uparrow} \qquad \frac{\vdash \Theta, P; \Gamma \Downarrow P}{\vdash \Theta, P; \Gamma \Uparrow} \qquad \frac{\vdash \Theta; \Gamma \Uparrow Q}{\vdash \Theta; \Gamma \Downarrow Q}$$

Figure 1.3: The focused proof-system for LL

Theorem 1.8 (Completeness of focusing for LL [And92])**.** $\vdash \Gamma$ *is provable in linear logic if and only if* $\vdash \Uparrow \Gamma$ *is provable in the focused calculus.*

Completeness was stated and proved by Andreoli. Several alternative proofs have been proposed since then. For example [MS07] and [Lau04] study focusing in more details as a proof transformation, the former one dealing with rule permutations in cut-free proofs while the latter shows that focusing can be obtained as a special case of cut-elimination.

A subtle feature of the focused system is its treatment of atoms. When focusing on a synchronous atom, the only possibility is to apply the axiom. For example, consider focusing on $(P \multimap a)^\perp$ where a is an asynchronous atom: the context is split and the focus remains on $a^\perp$ in the second premise, since it is synchronous the only way to complete that subderivation is the axiom, hence a must be already present in

the sequent. Hence, asynchronous atoms correspond to back-chaining — this observation can be used to recover uniform proofs [MNPS91] from focusing. Conversely, synchronous atoms yield a forward-chaining behavior.

Although focusing was motivated by backwards proof-search in linear logic, it is a fundamental property of linear logic in general. The focused calculus yields a big-step reading of formulas, which are not seen as made of connectives but rather of aggregates of connectives of the same nature, called *synthetic connectives*. For example, $A_0 \oplus (A_1 \oplus A_2)$ really becomes a ternary disjunction equipped with three projections. Going further, since asynchronous subformulas can be decomposed immediately after synchronous steps, one can aggregate a layer of synchrony with the next layer of asynchrony — the result is called a *bipole*. This big-step refinement of linear logic has, for example, a high performance impact in forward proof-search [Cha06]. At a more fundamental level, focusing also inspired Girard's ludics [Gir01] and participated in important new insights about the structure and dynamics of logic [Gir91a, Gir91b].

It is worth pointing out, however, that focusing in itself is a somehow weak observation. Its interest and weakness is that it only relies on a shallow syntactical criterion, which can be easily fooled: an asynchronous formula P can be turned into an equivalent synchronous one by forming $P \otimes \mathbf{1}$, and conversely by considering $P \parr \bot$. Such constructions are called *delays* since they have the effect to delay the treatment of the formula in proof-search. But the shallow syntactic notion of synchrony also turned out to be related to a deeper one, namely positivity.

Definition 1.9 (Positive,Negative)**.** A formula P is said to be positive if $P \equiv !P$, negative if $P \equiv ?P$. A connective is said to be positive (resp. negative) if it preserves positivity (resp. negativity).

Proposition 1.10. *In LL, synchronous (resp. asynchronous) connectives are exactly positive (resp. negative) connectives.*

We show only the non-trivial direction for the tensor case:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash P}\quad\dfrac{}{Q \vdash Q}}{P, Q \vdash P \otimes Q}}{!P, !Q \vdash P \otimes Q}}{!P, !Q \vdash !(P \otimes Q)} \qquad \dfrac{\dfrac{\dfrac{\textit{Assumed}}{P \vdash !P}\quad\dfrac{\textit{Assumed}}{Q \vdash !Q}}{P, Q \vdash !(P \otimes Q)}\ \textit{cut}}{P \otimes Q \vdash !(P \otimes Q)}$$

This characterization exploits the exponentials to force the order or rule applications (the negated tensor has to be introduced first), not so much for their non-linear behavior.

To conclude, let us add that focusing has been successfully adapted to intuitionistic [DJS93, DJS95, LM07a], classical [Lau02, LR03, LQdF05, LM07b] but also non-commutative logics [Hen93] for example. It is thus a major property of proof-theory in general. However, focusing is only so unambiguously principled in linear logic, which remains the most appropriate framework to start studying it.

1.5 Canonicity

An interesting question about a logic, and more precisely about its connectives, is whether they are uniquely determined. This might sound like a semantic question, but proof-theory is in fact sufficient for studying it.

Definition 1.11 (Canonicity)**.** A connective can be duplicated by adding another one of same type, and duplicating all the rules of the original connective into corresponding ones for its copy. In other words, theorems remain theorems when you change one version of the connective for the other. A connective is said to be *canonical* if it is equivalent to all of its duplicates.

Atoms are non-canonical *par excellence*: they are pure names without any logical structure. On the other hand, the conjunction is uniquely defined by its logical behavior. In LJ, suppose that we duplicate conjunction into $\cap$, hence adding the rules:

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \cap Q} \qquad \frac{\Gamma, P, Q \vdash R}{\Gamma, P \cap Q \vdash R}$$

It is then possible to derive the equivalence between $P \cap Q$ and $P \wedge Q$ for any P and Q:

$$\frac{\dfrac{\overline{P, Q \vdash P} \quad \overline{P, Q \vdash Q}}{P, Q \vdash P \wedge Q}}{P \cap Q \vdash P \wedge Q} \qquad \frac{\dfrac{\overline{P, Q \vdash P} \quad \overline{P, Q \vdash Q}}{P, Q \vdash P \cap Q}}{P \wedge Q \vdash P \cap Q}$$

In a sense, canonicity establishes the connection between the connective (the symbol) and its logical rule (the behavior).

In MALL, all connectives are canonical. Exponentials, however, are not. Let us copy $!, ?$ into $\hat{!}, \hat{?}$ and add notably the following rule:

$$\frac{\vdash \hat{?}\Gamma, P}{\vdash \hat{?}\Gamma, \hat{!}P}$$

There is no way to relate $?$ and $\hat{?}$. One can see here that non-canonicity is closely related to the syntactical "sanity check" of having only one connective involved per inference rule — except for those of the sequent structure. In that case, non-canonicity can be understood from the very meaning of exponentials: they provide infinite availability, and infinity is not an uniquely defined concept.

It is interesting to notice that exponentials do have a canonical focusing behavior: $\hat{?}$ would have to be asynchronous and $\hat{!}$ synchronous. Atoms are both non-canonical and do not have a canonical focusing behavior, while MALL connectives are both canonical and have a canonical focusing treatment. Anticipating the next chapters, the fixed point connectives μ and ν will be the opposite of exponentials: they have a canonical meaning but not a unique focusing treatment.

1.6 Terms and equality

Although first-order logic is standard and widely understood, equality is less frequently considered within proof-theory, and its treatment is subtle. We detail here our approach

to equality, which dates back to [Gir92, SH93]. Historically, this notion of equality is a byproduct of the introduction of fixed points. But it gains to be introduced separately.

What is clear about equality is its right rule: reflexivity. But there is no clear cut for the design of the left rule. We shall consider the following rules:

$$\frac{\{\Sigma\theta; \Gamma\theta \vdash G\theta : u\theta = v\theta\}}{\Sigma; \Gamma, u = v \vdash G}\ {=}L \qquad \frac{}{\Sigma; \Gamma \vdash u = u}\ {=}R$$

The left rule has one premise for each unifier θ of $u \doteq v$. The application of θ to terms is standard and naturally extended to formulas and to the left hand-side of the sequent. Its application to the signature ($\Sigma\theta$) denotes the signature obtained by removing from Σ the variables that are in the domain of θ, and adding those that are in its range.

In a sense, the left rule is the naive dual of the right one: it enumerates all cases for which the right one might have been proved. Indeed, this design supports cut-elimination. The principal case only consists in permuting $=R$ below the cut. The interesting phenomenon occur when reducing a cut on first-order quantifiers: this results in the instantiation of the universal variable by the witness of the existential quantification. That instantiation has to be performed in a subderivation, preserving its validity — this is a common simple result. The new case here is $=L$: as a variable gets instantiated, some unifiers might be simply updated, but others might disappear. If the equality eventually becomes absurd, the corresponding instance of $=L$ has no more premise. It is also easy to expand the axiom on equality, and along the same lines we obtain commutativity and canonicity of equality:

$$\frac{\ldots \quad \dfrac{}{\vdash u\theta = v\theta}\ {=}R \quad \ldots}{u = v \vdash u = v}\ {=}L$$

Infinitary rules are often convenient, but can be rightfully criticized. Indeed, a proof should always be finitely presentable, so that its validity can be decided. It is also a practical issue that proofs can be built in a finite amount of time. Hence, we usually consider a specialized version of the left rule, relying on a *complete set of unifiers* (*csu*), *i.e.,* a set S of unifiers such that all unifiers of $u \doteq v$ are specializations $\theta\theta'$ of some $\theta \in S$:

$$\frac{\{\Sigma\theta; \Gamma\theta \vdash G\theta : \theta \in csu(u \doteq v\theta)\}}{\Sigma; \Gamma, u = v \vdash G}$$

That rule is equivalent to the previous one: in one direction it is because the complete set of unifiers is a subset of all unifiers, in the other because the difference between the two can be obtained by specializing substitutions, and proofs accordingly. In the case of first-order terms, the *csu* can in fact be a most general unifier. However, that rule is still not effective in general in the case of higher-order terms; in practice it can often be managed by using higher-order pattern unification [Mil92].

Example 1.12. The *csu*-based rule is natural to work with, as it only requires the essential information. For example, with first-order terms:

$$\frac{\dfrac{}{x; Px \vdash Px}}{x, y; x = y, Px \vdash Py}$$

With higher-order terms, in the higher-order pattern fragment:

$$\dfrac{\dfrac{\vdots}{z; \vdash \exists z'.\ (\lambda a.\ z) = (\lambda a.\ z') \wedge (\lambda b.\ z) = (\lambda b.\ z')}}{x, y; (\lambda a \lambda b.\ x\ a) = (\lambda a \lambda b.\ y\ b) \vdash \exists z'.\ x = (\lambda a.\ z') \wedge y = (\lambda b.\ z')}$$

As is clear from these examples, universal variables are not constants. Hence, we avoid to call them eigenvariables as Gentzen [Gen69] did.

It is important to notice that the left equality rule does not enumerate possible instantiations, but really all possible substitutions. For example, if a type τ is empty in the current signature, there is no ground instantiation but there is still the identity substitution. In other words, the equality rule does not embody any reasoning on the signature. The same is true of the universal quantifier. Semantically speaking, $\forall x^\tau.\bot$ holds when τ is vacuous. However, its proof theoretic treatment is generic: the vacuity of τ is not an obstacle to extending the signature with a new universal variable. In fact, the only connective that takes Σ into account is the existential quantifier; still, it does not reason on it but only builds a witness from it. As a consequence, derivations are stable by extension of the signature. A left equality rule working on instantiations rather than unifiers would not be very practical and could not be reduced to unification. Moreover, it would simply be consistent with universal quantification, as there is no cut-free proof of the following:

$$\dfrac{\dfrac{\dfrac{}{x; x = x \vdash \bot} \quad \dfrac{}{x; \vdash x = x}}{x; \vdash \bot}}{\vdash \forall x^\tau.\ \bot}$$

Studying a finer-grained rule for equality might be of interest. By putting unification steps into the logic, one would avoid the need to rely on an external unification procedure to obtain the effective *csu*-based rule. This could also allow to move more easily from finite to possibly infinite terms, by removing the occur-check clauses. The advantage of that presentation is that it is light and convenient to work it, so it does not interfere with the main goal of this thesis.

Chapter 2

Fixed points

This chapter is dedicated to the introduction of fixed points, which are the central concept investigated in this thesis. Although some forms of fixed points appeared early in proof-theory, the general treatment of (co)induction should probably be attributed to Mendler's work in type theory [Men87]. In this chapter, we present a line of work within sequent calculus, starting in the '90 in the field of logic programming. This allows for a progressive introduction that illustrates clearly important proof-theoretical aspects of fixed points. More precisely, this introduction entirely lies in the intuitionistic setting. We do not intend to fully develop the proof-theoretical aspects encountered in that introduction: the useful ones will be detailed in the simpler proof-theoretical setting of μMALL (Chapter 3). Finally, we shall also compare our treatment of least and greatest fixed points with other treatments, namely in type theory and cyclic proofs.

2.1 From logic programming to fixed point definitions

A similar proof-theoretical treatment of fixed points was proposed independently by Girard [Gir92] and Schroeder-Heister [SH93]. In both cases, the goal was to renew the foundations of logic programming and in particular give a logical account of *negation-as-failure*.

In (pure) Prolog, one uses clauses to specify a program, for example:

$$\begin{array}{lll} \forall XL. & \mathit{mem}\ X\ (X :: L) & \\ \forall XYL. & \mathit{mem}\ X\ (Y :: L) & \subset \quad \mathit{mem}\ X\ L \end{array}$$

Given such a program $\mathcal{P}$, the user can run Prolog against a query G, *e.g., mem* 3 *nil*. Prolog then executes a goal directed, complete proof-search strategy on the sequent $\mathcal{P} \vdash G$. Such an approach is open-ended: if $\mathcal{P} \vdash G$ is provable, then so is $\mathcal{P}, \mathcal{P}' \vdash G$. Hence, the extension of a program can only add more facts. In other words, the logical foundation of Prolog only allows to inspect the consequences of a program, but not to reason about its causes, and in particular to establish negations.

An extra-logical argument, or meta-level observation, shows that Prolog is in fact finding inhabitants of an inductive specification. Indeed, it is building objects by a

finite chaining of the clauses of the logic program. The clauses of the program can be read as the clauses of an inductive specification, the inductive aspect being inherited from the inductive, finite structure of the derivations that Prolog builds. Formally, the representation of inductive specifications as logic programs is *adequate*: there is a bijection between the objects inductively specified and their representations, *i.e.,* the derivations built by Prolog. It is important to notice that this adequacy depends on the very directed proof-search strategy that Prolog applies. In general, sequent calculus proofs working on Horn clauses can be very much unfocused: one can partially instantiate a clause, then drop it, or interleave it with the another instantiation, etc. In order to obtain adequacy results, one has to consider *uniform proofs* [MNPS91] — which are a particular kind of focused proofs. In natural deduction, normal forms are enough for obtaining adequacy.

The notion of definition proposed in [Gir92, SH93] partially internalizes the meta-level observation that a logic program can be read as an inductive specification. Only partially, because the resulting notion will not define a least fixed point, *i.e.,* an inductive specification, but only an arbitrary fixed point. From the point of view of the user writing a specification, moving to definitions [Gir92, SH93] is a negligible syntactic shift. Our example becomes:

$$\begin{array}{lcl} \mathit{mem}\ X\ (X :: L) & \triangleq & \top \\ \mathit{mem}\ X\ (Y :: L) & \triangleq & \mathit{mem}\ X\ L \end{array}$$

The novelty lies in the proof-theoretical treatment of the definitions, which reflects a *closed-world* reading in a rich way. For doing so, the logic has to treat *defined atoms* as connectives. As such, they shall be equipped with introduction and elimination rules. The right rule expresses that definitions are derivable from their clauses, supporting *backchaining* in logic programming. The left rule is a *case-analysis* principle, expressing that *only* the clauses can be used to derive a defined atom.

$$\frac{\{\ (\Gamma, B \vdash P)\theta\ :\ A \stackrel{\triangle}{=} B \text{ and } A'\theta \doteq A\theta\ \}}{\Gamma, A' \vdash P} \qquad \frac{\Gamma \vdash B\theta \quad A \stackrel{\triangle}{=} B}{\Gamma \vdash A\theta}$$

Before anything else, it is important to notice that LJ extended with definitions still represents the same objects as before. In other words, the encoding of an inductive specification as a set of definitions is still adequate. It is in fact straightforward, without even having to restrict to focused derivations: with definitions, there is no more irrelevant information. In fact, as long as one considers what is basically a Horn clause specification, there is exactly one formula on the right hand-side.

So, intuitionistic logic extended with definitions allows for adequate representations. But it allows for much richer reasoning about the represented objects, thanks to the left rule on defined atoms — that interpretation of left rules being justified by cut-elimination. For example, one can prove that a given item is not in a given list. The case analysis rule allows to inspect *finite behavior*, *i.e.,* a finite part of the causes of an hypothesis. This brings a logical foundation to the notion of negation as (finite) failure in particular, and more generally to model-checking, which shall be exploited in the logic programming language Bedwyr (*cf.* Chapter 7).

In general, a finite case analysis may not suffice to prove a theorem. For example, $\forall x \forall l.\ mem\ x\ l \supset mem\ x\ l$ cannot be obtained by a finite exploration of the hypothesis, as there are infinitely many list inclusions. It shows that instances of the axiom rule on defined atoms cannot be eliminated in general. It can be expanded, but the iterated expansions may not terminate, as is the case in our example. In other words, while defined atoms have a logical behavior, they must still be treated as atoms.

2.1.1 Cut-elimination

The cut-elimination procedure is extended with a reduction for definitions, where the right definition rule selects a clause among all cases of the left definition rule:

$$\dfrac{\dfrac{\Sigma; \Gamma \vdash B\theta \quad A \stackrel{\triangle}{=} B \quad A' = A\theta}{\Sigma; \Gamma \vdash A'} \quad \dfrac{\{(\Sigma; \Gamma, B \vdash P)\theta \ :\ A \stackrel{\triangle}{=} B, A'\theta \doteq A\theta\}}{\Sigma; \Gamma, A' \vdash P}}{\Sigma; \Gamma \vdash P}$$

$$\downarrow$$

$$\dfrac{\Sigma; \Gamma \vdash B\theta \quad \Sigma; \Gamma, B\theta \vdash P}{\Sigma; \Gamma \vdash P}$$

In order to preserve consistency, definitions are required to be *monotonic*: there should not be a negative recursive occurrence of the defined atom in any of its defining clauses. Without this constraint, it is indeed easy to break consistency, or more precisely break the termination of cut-elimination. Let us define p to be $p \supset \bot$, and consider the following derivation, which we shall name Ω for a good reason:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p, \bot \vdash \bot}\ init \quad \dfrac{}{p \vdash p}\ init}{p, p \supset \bot \vdash \bot}\ \supset L}{p, p \vdash \bot}\ defL}{p \vdash \bot}\ contract}{\vdash p \supset \bot}\ \supset R}{\vdash p}\ defR$$

Now, cutting Ω against its subderivation establishing $p \vdash \bot$, we obtain a proof of $\bot$. One can check that on this derivation, the cut-elimination procedure loops. In fact, its behavior is that of the λ-term $(\lambda x.\ xx)$ applied to itself: it makes two copies of itself (which corresponds to the contraction rule above) and applies one to the other.

It is interesting to notice that nothing in the cut reduction rule requires monotonicity. The inconsistency is a global non-termination problem. As a matter of fact, Girard made explicit the relationship between structural rules, inconsistency and definitions. In [Gir92] he notes that in linear logic, negative occurrences of the defined atom were safe as long as they are not under any exponential. The reason is that the "inconsistent" features need exponentials to really go wrong. This idea has been pushed further by noting that light exponentials, which guarantee polynomial-time reduction, can also be used to force everything to terminate: this has been done with fixed points [Gir98] but also naive set theory [Ter04].

2.1.2 The self-dual μ combinator

The line of work on definitions continued, notably with the study of induction and coinduction [MM00, MT03b, Tiu04]. Before presenting these aspects, let us show how to move from the notion of defined atoms, where the logic is parametrized by a set of definitional clauses, to a self-contained formalism where defined atoms are replaced by μ expressions. That notation is used in several other formalisms, notably type theory [Mat99] and modal μ-calculus.

Although equality can be encoded as a defined atom, we take the converse approach and consider equality as an independent notion, as described in the previous chapter. This allows to rewrite all definitions in just one clause with flexible parameters in the head, for example:

$$mem\ X\ L \stackrel{\triangle}{=} (\exists L'.\ L = X :: L') \vee (\exists Y \exists L'.\ L = Y :: L' \wedge mem\ X\ L')$$

The definition in that form behaves as before: the toplevel disjunction allows to select one of the clauses on the right, and enumerates all possible clauses on the left; the equalities encode the unification constraints. The behavior of the definition rules is now restricted to pure unfoldings. We are ready to replace the notion of definition by a logical connective expressing this core concept.

For any term types $\vec{\gamma}$ we add a fixed point combinator $\mu_{\vec{\gamma}}$ of type $(\vec{\gamma} \to o) \to (\vec{\gamma} \to o)$. These new logical connectives are self-dual, and the axioms cannot in general be eliminated on them. In other words, we consider the following rules:

$$\frac{\Gamma, B(\mu B)\vec{t} \vdash G}{\Gamma, \mu B\vec{t} \vdash G} \qquad \frac{\Gamma \vdash B(\mu B)\vec{t}}{\Gamma \vdash \mu B\vec{t}} \qquad \frac{}{\Gamma, \mu B\vec{t} \vdash \mu B\vec{t}}$$

The obtained notion of general fixed point is sometimes called a *retract*.

As long as there is no mutual recursion, it is straightforward to translate a definition into a single clause and finally turn it into a fixed point, *i.e.,* a μ-formula. This is done by first translating the defined atoms occurring in the (single) clause defining the considered atom. More details about this translation are provided in Chapter 5 where we also study the case of mutually defined fixed points.

Example 2.1. We introduce the term type n for natural numbers, with two constants $0 : n$ and $s : n \to n$. We define *nat* of type $n \to o$ as a fixed point:

$$\begin{aligned} B_{nat} &\stackrel{def}{=} \lambda nat \lambda x.\ x = 0 \vee \exists y.\ x = s\ y \wedge nat\ y \\ nat &\stackrel{def}{=} \mu B_{nat} \end{aligned}$$

The second-order abstraction B_{nat} shall be called a *predicate operator*, or simply the *body* of the fixed point expression μB_{nat}. Notice that reasoning takes place on the fixed point formula *nat*, not on the type n which could as well contain other pointless constants.

Unsurprisingly, our new connective is not canonical. Indeed, the two rules given here hold for any fixed point, and there can be many in general. Let us consider the duplicate $\hat{\mu}$ equipped with the same inference rules as μ, and try to establish $\mu B_{nat} x \vdash$

$\hat{\mu}B_{nat}x$. The only possibility is to unfold the left hand-side fixed point, which yields two cases: x is either zero or the successor of some y. In the first case, we can prove $\vdash \hat{\mu}B_{nat}0$. But the second one is the same as our initial goal: $\mu B_{nat}y \vdash \hat{\mu}B_{nat}y$. This looping attempt calls for a proof by induction, which in turn requires capturing not only fixed points but more precisely least fixed points.

2.2 The logic μLJ

We present the logic μLJ, our intuitionistic system of reference supporting least and greatest fixed points. Its language of formulas is extended not only with the connective μ, now representing *least* fixed points, but also ν, of the same type, representing greatest fixed points. The rules of μLJ are presented in Figure 2.1. We present and discuss below the treatment of fixed points, starting with least fixed points.

2.2.1 Least fixed points

In proof-theory, least fixed points are characterized by the ability to reason about them by *induction*. It is interesting to justify that characterization, and its formalization in μLJ, from other presentations of least fixed points. It also shows that we are considering a natural notion, and not an exotic connective or an ad-hoc increment of expressiveness.

Definition 2.2 (Fixed, prefixed and postfixed point)**.** Let ϕ be a mapping from sets to sets[1]. The set S is said to be a *fixed point* of ϕ when $\phi(S) = S$; a *prefixed point* of ϕ when $\phi(S) \subseteq S$; a *postfixed point* of ϕ when $S \subseteq \phi(S)$.

Example 2.3. The predicate operator B_{nat} can be read as the following function:

$$N \mapsto \{0\} \cup \{s\ y : y \in N\}$$

Its prefixed points contain zero and are stable by successor. Its postfixed points do not necessarily contain zero, but each of their elements is either zero or the successor of another. It admits a least fixed point, obtained by iterating from the empty set: it is the usual set of natural numbers. The greatest fixed point, assuming that there exists objects x which are not natural numbers, would contain them as well as their successors $s^n x$; assuming infinite terms, the greatest fixed point would also contain the infinite chain of successors.

Theorem 2.4 (Knaster-Tarski)**.** *Let ϕ be a monotonic function, then ϕ has a least fixed point, which is the intersection of all its prefixed points.*

In category theory, least fixed points are initial algebras.

Definition 2.5 (Initial algebra)**.** Given an endofunctor F of the category $\mathbf{C}$, an F-algebra is an object A together with a morphism $\alpha : FA \to A$. An initial F-algebra

[1] One can more generally consider a mapping on a complete lattice, but we seek the most intuitive presentation.

Propositional intuitionistic logic

$$\frac{}{\Sigma; \Gamma, \bot \vdash P} \quad \frac{}{\Sigma; \Gamma \vdash \top}$$

$$\frac{\Sigma; \Gamma, P, P' \vdash Q}{\Sigma; \Gamma, P \wedge P' \vdash Q} \quad \frac{\Sigma; \Gamma \vdash P \quad \Sigma; \Gamma \vdash Q}{\Sigma; \Gamma \vdash P \wedge Q}$$

$$\frac{\Sigma; \Gamma, P_0 \vdash Q \quad \Sigma; \Gamma, P_1 \vdash Q}{\Sigma; \Gamma, P_0 \vee P_1 \vdash Q} \quad \frac{\Sigma; \Gamma \vdash P_i}{\Sigma; \Gamma \vdash P_0 \vee P_1}$$

$$\frac{\Sigma; \Gamma \vdash P \quad \Sigma; \Gamma, P' \vdash Q}{\Sigma; \Gamma, P \supset P' \vdash Q} \quad \frac{\Sigma; \Gamma, P \vdash Q}{\Sigma; \Gamma \vdash P \supset Q}$$

First-order structure

$$\frac{\Sigma, x; \Gamma, Px \vdash Q}{\Sigma; \Gamma, \exists x.Px \vdash Q} \quad \frac{\Sigma; \Gamma \vdash Pt}{\Sigma; \Gamma \vdash \exists x.Px}$$

$$\frac{\Sigma; \Gamma, Pt \vdash Q}{\Sigma; \Gamma, \forall x.Px \vdash Q} \quad \frac{\Sigma, x; \Gamma \vdash Px}{\Sigma; \Gamma \vdash \forall x.Px}$$

$$\frac{\{(\Sigma; \Gamma \vdash Q)\theta : t\theta \doteq t'\theta\}}{\Sigma; \Gamma, t = t' \vdash Q} \quad \frac{}{\Sigma; \Gamma \vdash t = t}$$

Fixed points

$$\frac{\Sigma; \Gamma, S\,t \vdash P \quad x; BS\,x \vdash S\,x}{\Sigma; \Gamma, \mu B t \vdash P} \quad \frac{\Sigma; \Gamma \vdash B(\mu B)t}{\Sigma; \Gamma \vdash \mu B t}$$

$$\frac{\Sigma; \Gamma, B(\nu B)t \vdash P}{\Sigma; \Gamma, \nu B t \vdash P} \quad \frac{\Sigma; \Gamma \vdash S\,t \quad x; S\,x \vdash BS\,x}{\Sigma; \Gamma \vdash \nu B t}$$

Identity group

$$\frac{}{\Sigma; \Gamma, \mu B \vec{t} \vdash \mu B \vec{t}} \quad \frac{}{\Sigma; \Gamma, \nu B \vec{t} \vdash \nu B \vec{t}}$$

Figure 2.1: Inference rules for μLJ

(A, α) satisfies the following diagram for any F-algebra (B, β):

$$\begin{array}{ccc} FA & \xrightarrow{\alpha} & A \\ {\scriptstyle F(f)}\,\vdots & & \vdots\,{\scriptstyle f} \\ \downarrow & & \downarrow \\ FB & \xrightarrow{\beta} & B \end{array}$$

The Knaster-Tarski theorem gives us an induction rule, along the common interpretation of implication as an inclusion:

"If $B(S) \subseteq S$ and $t \in \mu B$ then $t \in S$."

$$\frac{x; BS\,\vec{x} \vdash S\,\vec{x}}{\Sigma; \mu B\vec{t} \vdash S\,\vec{t}}$$

We obtain the same rule by interpreting the categorical notion of initial algebra, this time reading implication as the existence of a morphism: "If $FS \to S$ then $\mu F \to S$". The rest of the diagram could be identified in the cut reductions. The monotonicity condition of the Knaster-Tarski theorem, ensuring the existence of a (least) fixed point, translates in μLJ to the constraint that fixed point bodies are monotonic.

Example 2.6. In the particular case of *nat*, the above induction rule yields the usual induction principle:

$$\dfrac{\dfrac{\vdash P\,0 \quad Py \vdash P(s\,y)}{(B_{nat}P)x \vdash Px}\,\vee L, \exists L, =L}{nat\ x \vdash Px}$$

The problem with the considered induction rule is that it does not satisfy cut-elimination. We shall see that there is a way to reduce a cut between derivations of $\mu B \vdash S$ and $\vdash \mu B$, obtaining a derivation of the invariant S. But it is impossible to reduce a cut between derivations of $\mu B \vdash S$ and $S \vdash P$, until the invariant becomes active in the former derivation. In other words, the reduction of a cut on the invariant has to be postponed until a cut is reduced on the associated least fixed point. To express this, we consider in μLJ the following left rule for μ, which aggregates the former induction rule with a cut on the invariant, thereby restoring cut-eliminability:

$$\frac{\Sigma; \Gamma, S\,\vec{t} \vdash P \quad x; BS\,\vec{x} \vdash S\,\vec{x}}{\Sigma; \Gamma, \mu B\vec{t} \vdash P}$$

As shown in Figure 2.1, the right rule for μ is unchanged, and the axiom on least fixed points is necessary. (As before, notice that although our logic has become much more expressive with the introduction of the induction rule, it still represents the same objects.) There is no need to consider a left unfolding rule for μ. Indeed, induction can emulate unfolding in the case of a monotonic fixed point B, by picking the invariant $B(\mu B)$. This is detailed for LINC in [Tiu04] and in the next section for the linear case, which does not essentially differ.

As announced above, the connective μ is now canonical. Let us consider its duplicate $\hat{\mu}$, equipped with the same rules:

$$\frac{\Gamma, S\,\vec{t} \vdash G \quad BS\,\vec{x} \vdash S\,\vec{x}}{\Gamma, \hat{\mu}B\vec{t} \vdash G} \qquad \frac{\Gamma \vdash B(\hat{\mu}B)\vec{t}}{\Gamma \vdash \hat{\mu}B\vec{t}} \qquad \frac{}{\Gamma, \hat{\mu}B\vec{t} \vdash \hat{\mu}B\vec{t}}$$

We can prove the equivalence between μB and $\hat{\mu} B$, independently of B, using the (admissible) generalized axiom *init*:

$$\dfrac{\dfrac{}{\hat{\mu}B\vec{t} \vdash \hat{\mu}B\vec{t}} \quad \dfrac{\dfrac{}{B(\hat{\mu}B)\vec{x} \vdash B(\hat{\mu}B)\vec{x}}\,init}{B(\hat{\mu}B)\vec{x} \vdash \hat{\mu}B\vec{x}}\,\hat{\mu}R}{\mu B\vec{t} \vdash \hat{\mu}B\vec{t}}\,\mu L \qquad \dfrac{\dfrac{}{\mu B\vec{t} \vdash \mu B\vec{t}} \quad \dfrac{\dfrac{}{B(\mu B)\vec{x} \vdash B(\mu B)\vec{x}}\,init}{B(\mu B)\vec{x} \vdash \mu B\vec{x}}\,\mu R}{\hat{\mu}B\vec{t} \vdash \mu B\vec{t}}\,\hat{\mu}L$$

Once can check that the greatest fixed point combinator ν, and thus all connectives of μLJ, are also canonical. Since we do not consider predicate constants in that logic, it means that all formulas have a defined behavior. This is an important aspect of the systems that we consider in this thesis; it will also hold for μMALL (Chapter 3). Although most results (*e.g.,* cut-elimination, focusing) extend trivially to predicate constants, some do not: for example, we shall derive structural rules from the behavior of formulas in μMALL, and similarly derive generic structural rules in μLJ (Chapter 6). When such techniques are used, it is necessary to exclude undefined atoms.

2.2.2 Cut-elimination

We have introduced the proof-theoretical treatment of least fixed points in μLJ, by means of semantic intuitions. Although these intuitions are useful, and connections certainly exists, they do not need to be formally established to validate the design of μLJ: the syntactic, internal process of cut-elimination suffices. And a presentation of some of its key points should help understanding the logic.

The principal cut reduction for least fixed points is based on the transformation of a derivation of μB into a derivation of one of its invariants. Given a formula S and a proof Θ of $\forall \vec{x}.\, BS\,\vec{x} \supset S\,\vec{x}$, one can transform a derivation of $\Gamma \vdash \mu B\vec{t}$ into one of $\Gamma \vdash S\,\vec{t}$. This is done by induction on the derivation of the least fixed point, along the following scheme:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma' \vdash B(\mu B)\vec{t'}}}{\Gamma' \vdash \mu B\vec{t'}}}{\dfrac{\dfrac{\vdots}{\Gamma \vdash B(\mu B)\vec{t}}}{\Gamma \vdash \mu B\vec{t}}} \quad \rightarrow \quad \dfrac{\dfrac{\vdots}{\Gamma' \vdash BS\,\vec{t'}} \quad \dfrac{\Theta(\vec{t'})}{BS\,\vec{t'} \vdash S\,\vec{t'}}}{\Gamma' \vdash S\,\vec{t'}} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash BS\,\vec{t}} \qquad \dfrac{\Theta(\vec{t})}{BS\,\vec{t} \vdash S\,\vec{t}}}{\Gamma \vdash S\,\vec{t}}$$

The big steps here, represented by dots, consist in traversing the structure of B. For doing so it is crucial that B is positive. If it is strictly positive, then there is no recursive occurrence of μB that will ever occur on the left. Otherwise, some instances may occur on the left, but negatively. In any case these sub-formulas will only occur at toplevel on the right hand-side of the sequent, and can thus only be active in the right unfolding rule. This is essential, as it is the only thing that S can simulate. So, unlike the cut reduction for the self-dual μ, the reduction associated to least fixed point does rely on monotonicity.

It is in fact possible to refine the transformation, and obtain a more precise constraint on fixed points. The traversal of B can be expressed as a functoriality property, *i.e.,* the following rule:

$$\frac{\vec{x}; P\vec{x} \vdash Q\vec{x}}{\Sigma; BP \vdash BQ}\ functo$$

Such a rule is admissible if B is monotonic. But it might also be derivable for some negative B, for example $\lambda p.\ p \supset p$ without even using the premise $P\vec{x} \vdash Q\vec{x}$. The use of such *monotonicity witnesses* was investigated by Matthes [Mat99] in an extension of System F with fixed points. Matthes uses the term "positive" for what we called monotonic bodies, and "monotonic" for fixed points that have such monotonicity witnesses. This is more usual and precise, but we avoid to use the term "positive" for that matter, as it shall be heavily used to denote a polarity (*cf.* Definition 1.9). Moreover, we really only consider positive fixed points in this thesis.

Assuming only the functoriality of B, we can fully formulate the reduction of a principal cut on least fixed points:

$$\dfrac{\dfrac{\dfrac{\Pi}{\Gamma \vdash B(\mu B)\vec{t}}}{\Gamma \vdash \mu B\vec{t}}\,\mu R \qquad \dfrac{\dfrac{\Pi'}{\Delta, S\vec{t} \vdash G} \qquad \dfrac{\Theta}{BS\vec{x} \vdash S\vec{x}}}{\Delta, \mu B\vec{t} \vdash G}\,\mu L}{\Gamma, \Delta \vdash G}\,cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Pi}{\Gamma \vdash B(\mu B)\vec{t}} \qquad \dfrac{\dfrac{\dfrac{}{S\vec{x} \vdash S\vec{x}} \qquad \dfrac{\Theta}{BS\vec{y} \vdash S\vec{y}}}{\mu B\vec{x} \vdash S\vec{x}}\,\mu L}{B(\mu B)\vec{t} \vdash BS\vec{t}}\,functo \qquad \dfrac{\Theta(\vec{t})}{BS\vec{t} \vdash S\vec{t}}}{B(\mu B)\vec{t} \vdash S\vec{t}}\,cut}{\Gamma \vdash S\vec{t}} \qquad \dfrac{\Pi'}{\Delta, S\vec{t} \vdash G}}{\Gamma, \Delta \vdash G}\,cut$$

Identifying the functoriality property allows for an elegant presentation of the rule, and has proved to help structuring normalization proofs. We do not show a normalization proof for μLJ in this thesis. For cut-elimination proof in sequent calculus, we refer the reader to the work on LINC [MT03b, Tiu04], which is closely related to μLJ. The main difference is that it deals with defined atoms rather than fixed points, and limits definitions to be stratified: from the point of view of fixed points, it imposes strict positivity and forbids mutually[2] (co)inductive fixed points. Tiu conjectures [Tiu04] that monotonicity is enough for ensuring cut-elimination. This is backed by Matthes' proof for extensions of System F (a much stronger system than μLJ, by the way) and our proof of cut-elimination for μMALL, which both require only monotonicity.

Moreover, the explicit monotonicity witness opens the possibility to work safely with definitions that are not obviously well-founded. Such cases arise in practice, when a definition is not syntactically well-founded, but a parameter is decreasing in each

[2]Mutually inductive fixed points are sometimes called interleaved fixed points (by opposition to nested ones), because of the pattern formed by the introduction and usage of predicate variables in such fixed point formulas, *e.g.,* in arbitrarily branching trees: $\mu Tree.\ \top \vee Leaf \vee (\mu List.\ \top \vee Tree \wedge List)$.

non-positive recursive occurrence. It is the case, for example, of Tait's reducibility predicate, used to prove normalization of various calculi: it is defined by induction on the type. It might be useful to consider systems where such fixed points are allowed, and justified before any use by asking the user to prove the well-foundedness of the notion. We leave these considerations to further work.

2.2.3 Greatest fixed points

In many settings, least and greatest fixed points are duals of each other: in a complete lattice, reversing the order swaps least and greatest fixed points; this amounts to consider complements in set theory, in other words the complement of a least fixed point is given by the greatest fixed point of the dual operator; this is also observed in category theory [CS02]. Unsurprisingly, the treatment of greatest fixed points in μLJ is obtained by dualizing the rules for least fixed points — this will be even clearer in the next chapter, with the logic μMALL which internalizes this duality. The same observations can be made: admissibility of the right unfolding, canonicity, cut reductions, etc.

2.3 Comparison with related deductive principles

Before concluding, we quickly compare our framework with two interesting relatives: type theories supporting (co)inductive types, and cyclic proofs. We do not consider infinitary approaches such as infinitely deep derivations, or the infinitely wide ω rule which enumerates all natural numbers, and by extension all inhabitants of a fixed point. Such systems belong more to the model-theoretic approach to logic, and do not constitute valid notions of proofs to us. Moreover, these treatments tend to hide the distinction between least and greatest fixed points by pushing it to the meta-level, while we are trying to analyze it precisely.

2.3.1 Type theory

Inductive and coinductive types have been considered in various type theories [Men87, PM96, Gim96, Gim98]. We consider in particular the Calculus of Inductive Constructions used in Coq. That system is much more powerful than μLJ, which is only a first-order logic. But it is interesting to relate the treatment of (co)inductive types in both systems, even informally, for the types that can be expressed in both. For example, what is the difference between an induction on *nat* in Coq and μLJ?

In Coq, both inductive and coinductive types can be eliminated by pattern matching: from the logical point-of-view this provides finite case-analysis. Conversely, finite constructions of (co)inductive types is provided by the associated constructors. Inductive types have a special eliminator, called `fix`, which allows to build a recursive function over an inductive type, under the condition that all recursive calls of the function are done on a strict subterm of the argument that is inducted over. Intuitively, this ensures that the function terminates because an inhabitant of an inductive type is only made of a finite number of constructors. Conversely, coinductive types have `cofix`,

which allows to build infinite objects recursively, under the condition that recursive occurrences are under constructors. Intuitively, this ensures that any finite inspection of the coinductive construction will terminate, since each recursive call brings some new information.

The induction rule of μLJ can be seen as a combination of `fix` and (shallow) pattern matching on the inductive argument. The invariance condition $BS\,\vec{x} \supset S\,\vec{x}$ corresponds to the guard of `fix`, in a more restrictive version: it requires that, assuming that S holds for immediate predecessors of $\vec{x}$, it holds for $\vec{x}$. It is more restrictive because it only provides recursive instances of S for the immediate predecessor, while `fix` allows recursive calls to arbitrary predecessors. A typical way to obtain such a strong induction principle is to use the accessibility relationship. For example, accessibility for *nat* corresponds to the relation $\geq$:

$$acc \stackrel{def}{=} \mu A \lambda x \lambda y.\ x = y \vee \exists x'.\ x = s\ x' \wedge A\ x'\ y$$

Then, the strong induction is obtained by taking invariants of the form $\lambda x.\ \forall y.\ acc\ x\ y \supset P\ y$. However, the accessibility relationship cannot always be defined in μLJ, for example, with the purely propositional version of *nat*, *i.e.,* $\mu X.\ \top \vee X$. A finer solution can be considered even in those cases. Although an inductively defined function may proceed to recursive calls on arbitrary subterms, there is a bound k to that depth, fixed by the patterns used to extract subterms. To obtain a similar behavior in μLJ, it suffices to consider the (less) strengthened invariant $S \wedge BS \wedge \ldots \wedge B^{k-1}S$.

The discussion is the same for the treatment of greatest fixed points.

2.3.2 Cyclic proofs

The induction and coinduction rules of μLJ obviously complicate its proof-theory, the most immediate consequence being that the subformula property does not hold anymore. An appealing way to recover it, while keeping finitely representable derivations, is to consider cyclic proofs where fixed points are only unfolded and the (co)inductive nature of a fixed point determines the conditions under which a cycle may be formed.

Luigi Santocanale [San02] proposed a calculus of circular proofs for a very simple logic, without implication nor quantifiers, which strikingly enjoys cut-elimination. Brotherston and Simpson [Bro06, BS07] developed a rich cyclic proof system for classical[3] first-order logic, which however does not satisfy cut-elimination. For the rest of this section, we make a couple observations on a similar system for intuitionistic logic.

We show an example of cyclic proof, with cut, where N is the natural inductive

[3]The classical nature of their system allows for striking examples of cut-free cyclic proofs, which would not be possible in an intuitionistic setting. This is the case, for example, of $\forall x.\ nat\ x \supset even\ x \vee odd\ x$. However, other theorems such as the totality of the addition fail to be proved even in the classical system.

definition of natural numbers, and E that of even natural numbers:

$$\dfrac{\dfrac{\dfrac{}{\vdash E\ 0}}{\vdash E\ 0 \vee E\ (s\ 0)} \quad \dfrac{\text{loop to (*)}}{N\ y \vdash E\ y \vee E\ (s\ y)} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{E\ y \vdash E\ y}}{E\ y \vdash E\ (s^2\ y)}}{E\ y \vdash E\ (s\ y) \vee E\ (s^2\ y)} \quad \dfrac{\dfrac{}{E\ (s\ y) \vdash E\ (s\ y)}}{E\ (s\ y) \vdash E\ (s\ y) \vee E\ (s^2\ y)}}{E\ y \vee E\ (s\ y) \vdash E\ (s\ y) \vee E\ (s^2\ y)}\ \mathit{cut}}{N\ y \vdash E\ (s\ y) \vee E\ (s^2\ y)}}{(*)\ N\ x \vdash E\ x \vee E\ (s\ x)}$$

The loop in that proof is valid because the left hand-side inductive is unfolded at each traversal of the loop. Brotherston actually considered very complex looping schemes, including cross-branches loops, and developed a general soundness criterion.

It is easy to see that there is no cut-free cyclic derivation establishing that every natural number is either even or odd. Assume that there is a cut-free proof of $\mathit{nat}\ x \vdash \mathit{even}\ (s^n x) \vee \mathit{odd}\ (s^n x)$ for some n. It cannot start with a $\vee R_i$ because the resulting subgoal would be invalid, hence it must do a case-analysis on $\mathit{nat}\ x$. The zero case is provable, but the premise for the successor case is $\mathit{nat}\ y \vdash \mathit{even}\ (s^{n+1} y) \vee \mathit{odd}\ (s^{n+1} y)$. That subderivation cannot start with a loop rule[4], hence we have a smaller derivation for $n + 1$. By infinite descent on the size of the derivation, this is absurd.

Intuitively, invariants can be obtained from cyclic proofs, extracted from sequents on which loops are formed — in our example, the invariant is $\lambda x.\ E\ x \vee E\ (s\ x)$. It seems possible to translate cyclic proofs into μLJ, where the cuts can then be eliminated. This is illustrated for our example:

$$\dfrac{\mathit{Id}\quad \dfrac{\dfrac{\dfrac{}{\vdash E\ 0}}{\vdash E\ 0 \vee E\ (s\ 0)} \quad \dfrac{}{E\ y \vee E\ (s\ y) \vdash E\ y \vee E\ (s\ y)} \quad \dfrac{\vdots}{E\ y \vee E\ (s\ y) \vdash E\ (s\ y) \vee E\ (s^2\ y)}}{E\ y \vee E\ (s\ y) \vdash E\ (s\ y) \vee E\ (s^2\ y)}\ \mathit{cut}}{B(\lambda x.\ E\ x \vee E\ (s\ x))x \vdash E\ x \vee E\ (s\ x)}}{N\ x \vdash E\ x \vee E\ (s\ x)}$$

However, there is an extra difficulty: cyclic proofs usually allow cycles going back to arbitrary predecessors, not only immediate ones. This difference can be overcome as explained in the previous section.

Compared to derivations using explicit (co)induction rules as in μLJ, cyclic proofs have two weaknesses: they do not allow to generalize a goal in order to make it invariant; and they do not allow reasoning about the invariant obtained for the predecessor by having it explicitly in the sequent. When applying cut-elimination to cyclic proofs, there are precisely two places where cuts cannot be reduced, corresponding to these two limitations: openings and closings of loops.

Even if cut-free cyclic proof systems are far from being complete, they are a natural restriction to consider from an implementation point of view, since they can easily be implemented through tabling [Pie05]. We experimented with that approach, for example, in the Bedwyr system (Chapter 7). From a more theoretical point of view,

[4]Of course, the possibility of generalizing the patterns allowed when looping is bound to fail with a more complex example, and would only make things obscure.

one may look for fragments of a logic with explicit (co)induction rules for which some kind of cyclic proof system is complete. Chapter 5 of this thesis will provide some answers in that direction.

Chapter 3

The logic μMALL

We present the logic μMALL which is at the core of our work, in itself and as a basis for carrying observations to other systems. It is obtained by first extending the multiplicative and additive fragment of linear logic (MALL) with equality and first-order quantification. Because of the bounded use of formulas during proof construction, provability in first-order MALL can be reduced to deciding unification problems under a mixed quantifier prefix. That is decidable if terms are first-order, although not if terms are simply typed λ-terms. An elegant and well-known way to make this logic more expressive is to add the exponentials ! and ? and the inference rules that allow for certain occurrences of formulas marked with these to be contracted and weakened [Gir87]. Such modal-like operators are not, however, without their problems. In particular, there is not an unique way to formulate the rules of the exponentials (see for example elementary and light linear logic [Gir98] and soft linear logic [Laf04]). Even if we fix the inference rules for the exponentials, as in standard linear logic, we have seen in Chapter 1 that the defined connectives are not canonical. It is certainly possible to consider a (partially ordered) collection of identically behaved exponentials on top of MALL (see for example [DJS93]).

An alternative to strengthen MALL with exponentials is to extend it with fixed points. Early approaches to adding fixed points [Gir92, SH93] involved inference rules that could only unfold fixed point descriptions: as a consequence, such logics could not discriminate between a least and greatest fixed point. Stronger systems that allow induction [MM00] as well as co-induction [Tiu04, MT03b] include inference rules using a higher-order variable that ranges over prefixed or postfixed points (invariants or coinvariants). We shall explore this alternative to exponentials: we extend first-order MALL to μMALL by adding least and greatest fixed point constructions. The obtained system gives a structured approach to infinity, and turns out to be very expressive.

Besides considering fixed points as alternatives to the exponentials, there are other reasons for examining μMALL. First, least and greatest fixed points are de Morgan duals of one another and, hence, the classical nature of linear logic should offer some economy and elegance in developing their proof theory, in contrast to intuitionistic logic. Second, since linear logic can be seen as the logic behind intuitionistic logic, it will be rather easy to develop a focusing proof system for intuitionistic logic and fixed

points based on the structure of the one we develop for μMALL.

It is important to stress that we are using linear logic here as "the logic behind computational logic" and not, as it is more traditionally understood, as the logic of resource management (in the sense of multiset rewriting, database updates, Petri nets, etc). Instead, we find the proof theory of linear logic an appropriate and powerful setting for exploring the structure of proofs in various intuitionistic logics (see [LM07a] for another such use of linear logic).

The rest of this chapter is organized as follows. In Section 3.1, we define μMALL, illustrate it through simple examples, and show some of its basic proof theoretic properties. We prove cut-elimination for μMALL in Section 3.2. Then, in Section 3.3, we establish a central property of μMALL's connectives that extends the classification of MALL's connectives between positive and negative ones. This observation highlights the expressivity of μMALL, which is shown through some examples in Section 3.4, and finally by encoding primitive recursive functions in Section 3.5.

3.1 Definition

In the following, terms are denoted by s, t; vectors of terms are denoted by $\vec{s}, \vec{t}$; formulas (objects of type o) are denoted by P, Q; variables are denoted by x, y. Finally, the syntactic variable B represents a formula abstracted over a predicate and n terms $(\lambda p \lambda x_1 \dots \lambda x_n . P p x_1 \dots x_n)$. We have the following formula constructors:

$$\begin{array}{rcl} P & ::= & P \otimes P \mid P \oplus P \mid P \parr P \mid P \,\&\, P \mid \mathbf{1} \mid \mathbf{0} \mid \bot \mid \top \\ & \mid & \exists_\gamma x.\, P \mid \forall_\gamma x.\, P \mid s \stackrel{\gamma}{=} t \mid s \stackrel{\gamma}{\neq} t \mid \mu_{\gamma_1 \dots \gamma_n}(\lambda p \lambda \vec{x}.\, P)\vec{t} \mid \nu_{\gamma_1 \dots \gamma_n}(\lambda p \lambda \vec{x}.\, P)\vec{t} \end{array}$$

The syntactic variable γ represents a term type and ranges over all simple types that do not contain o. The quantifiers have type $(\gamma \to o) \to o$ and the equality and *disequality* (*i.e.,* $\neq$) have type $\gamma \to \gamma \to o$. The connectives μ and ν have type $(\tau \to \tau) \to \tau$ where τ is $\gamma_1 \to \dots \to \gamma_n \to o$ for some arity $n \geq 0$. We shall almost always elide the references to γ, assuming that they can be determined from context when it is important to know their value. Formulas with top-level connective μ or ν are called fixed point expressions and can be arbitrarily nested. The first argument of a fixed point expression is called its *body*, and shall be denoted by B. In themselves, such second-order expressions are called *predicate operator* expressions or simply operators.

Quantifiers and (in)equality are not new and play a small role in the proof theory results: they are, however, crucial for our example applications. The central feature here is the fixed point constructs. Finally, note that there are no atoms (predicate constants) in the μMALL grammar. We shall see in the following the advantages of using fixed points instead.

Definition 3.1. We define the *negation* $\overline{B}$ of an operator expression B, and extend the usual definition of the involutive *negation* (*cf.* Definition 1.5) as follows:

$$\overline{B} \stackrel{def}{=} \lambda p.\lambda \vec{x}.(B(\lambda \vec{x}.(p\vec{x})^\perp)\vec{x})^\perp \qquad (s = t)^\perp \stackrel{def}{=} s \neq t \qquad (\mu B \vec{t})^\perp \stackrel{def}{=} \nu \overline{B} \vec{t}$$

An operator B is said to be *monotonic* when for any predicate variable p and terms $\vec{t}$, the negation normal and λ-normal form of $Bp\vec{t}$ does not contain any negated instance of p.

We shall assume that *all bodies are monotonic*. In other words, negation ($\bullet^\perp$ for formulas and $\overline{\bullet}$ for bodies) is not part of the syntax since negation normal form of formulas and bodies without atoms do not contain negations and since we forbid them explicitly in fixed point expressions. When we write negation in some inference rules, we shall be considering it as implicitly computing the negation normal form.

The monotonicity of a function is also a natural condition for the existence of fixed points in lattices or other models. The condition of monotonicity is used only syntactically here since we are not studying the semantics of μMALL.

We present the inference rules for μMALL in Figure 3.1. The initial rule is restricted to fixed points. In the ν rule, which provides both induction and coinduction, S is called the (co)invariant, and is a closed formula of the same type as νB, of the form $\gamma_1 \to \cdots \to \gamma_n \to o$. The treatment of equality[1] dates back to [Gir92, SH93]. In the disequality rule, *csu* stands for complete set of unifiers. This set has at most one element in the first-order case, but can be infinite in the presence of higher-order term variables, which we do not exclude. In that case, the proofs are infinitely branching but still have a finite depth. They are handled easily in our proofs by means of transfinite inductions. Again, the use of higher-order terms, and even the presence of the equality connectives are not essential to this work. All the results presented below hold in the logic without equality, and they do not make much assumptions on the language of terms.

Example 3.2. As an example we shall show why induction and coinduction coincide in the classical framework of linear logic. Let us consider the result of applying the ν rule on the negation of a least fixed point:

$$\frac{\vdash S\vec{t}, \Gamma \quad \vdash (S\vec{x})^\perp, \overline{B}S\vec{x}}{\vdash (\mu B\vec{t})^\perp, \Gamma}$$

Now, let S' be $S^\perp$. Then $\overline{B}S$ is simply $(BS')^\perp$ and we observe the usual induction rule in its one-sided form:

$$\frac{\vdash (S'\vec{t})^\perp, \Gamma \quad \vdash (BS'\vec{x})^\perp, S'\vec{x}}{\vdash (\mu B\vec{t})^\perp, \Gamma}$$

Finally, since negation is an involution, the rule holds for any invariant.

Proposition 3.3. *Let θ be a substitution. If $\Sigma; \vdash \Gamma$ is provable, then so is $\Sigma\theta; \vdash \Gamma\theta$.*

Proof. This is a standard property. It is established by induction on the proof, the case of equality being the only non-trivial one. When the conclusion of that rule is modified by the application of θ, the solutions θ' of the original unification problem are affected. If some θ' is incompatible with θ, then the corresponding branch does not have to be considered anymore. Otherwise, a branch should be considered for the composition of the two substitutions, but this is precisely the result of applying θ to the subderivation originally associated to θ'. The resulting set of unifiers is still a complete one. □

[1] See Chapter 1 for a detailed discussion about it.

MALL rules

$$\frac{}{\vdash \mathbf{1}} \quad \frac{\vdash \Gamma, P \quad \vdash \Delta, Q}{\vdash \Gamma, \Delta, P \otimes Q} \quad \frac{\vdash \Gamma, P, Q}{\vdash \Gamma, P \parr Q} \quad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}$$

$$\frac{}{\vdash \Delta, \top} \quad \frac{\vdash \Gamma, P \quad \vdash \Gamma, Q}{\vdash \Gamma, P \,\&\, Q} \quad \frac{\vdash \Gamma, P_i}{\vdash \Gamma, P_0 \oplus P_1}$$

First-order structure

$$\frac{\vdash \Gamma, Pt}{\vdash \Gamma, \exists x.Px} \quad \frac{\Sigma, x; \vdash \Gamma, Px}{\Sigma; \vdash \Gamma, \forall x.Px}$$

$$\frac{}{\vdash t = t} \quad \frac{\{\vdash \Gamma\theta : \theta \in csu(s \doteq t)\}}{\vdash \Gamma, s \neq t}$$

Fixed points

$$\frac{\vdash \Gamma, B(\mu B)\vec{t}}{\vdash \Gamma, \mu B\vec{t}}\ \mu \qquad \frac{\Sigma; \vdash \Gamma, S\vec{t} \quad \vec{x}; \vdash BS\vec{x}, (S\vec{x})^{\perp}}{\Sigma; \vdash \Gamma, \nu B\vec{t}}\ \nu \qquad \frac{}{\vdash \mu B\vec{t}, \overline{\nu B}\vec{t}}\ \mu\nu$$

Figure 3.1: Inference rules for μMALL

In the following, we write two sided sequents for convenience. We recall that $\Gamma \vdash \Delta$ is just a shorthand for $\vdash \Gamma^{\perp}, \Delta$.

Proposition 3.4. *Let P and Q be formulas (abstracted over some variables) and B a monotonic operator. The following rule, called functoriality of B, is admissible:*

$$\frac{\vec{x}; P\vec{x} \vdash Q\vec{x}}{\Sigma; BP \vdash BQ}\ functo$$

Proof. Let Π be a derivation of $\vec{x}; P\vec{x} \vdash Q\vec{x}$. The idea of the proof is to build expansions of Π. We proceed by induction on the number of fixed point connectives surrounding p in $Bp\vec{x}$, with a subinduction on the size of B. The interesting base case is when B is of the form $\lambda p.\ p\vec{t}$, in which we conclude by instantiating Π. The monotonicity of B excludes $\lambda p.\ (p\vec{t})^{\perp}$. The inductive steps for first-order MALL units and connectives are standard, for example the tensor is expanded as follows:

$$\frac{\dfrac{\dfrac{\vdots}{B_1P \vdash B_1Q} \quad \dfrac{\vdots}{B_2P \vdash B_2Q}}{B_1P, B_2P \vdash B_1Q \otimes B_2Q}}{B_1P \otimes B_2P \vdash B_1Q \otimes B_2Q}$$

And the two subderivations are respectively obtained by induction hypothesis on B_1 and B_2. The new cases are fixed points. When B is of the form $\lambda p.\ \nu(B'p)\vec{t}$, we proceed

as follows:

$$\dfrac{\dfrac{}{\nu(B'P)\vec{t} \vdash \nu(B'P)\vec{t}} \quad \dfrac{\dfrac{\vdots}{B'P(\nu(B'P))\vec{x} \vdash B'Q\nu(B'P)\vec{x}}}{\nu(B'P)\vec{x} \vdash B'Q\nu(B'P)\vec{x}}}{\nu(B'P)\vec{t} \vdash \nu(B'Q)\vec{t}}$$

The derivation is then completed by induction hypothesis on $\lambda p.\ B'p(\nu(B'P))\vec{x}$. The case of μ is treated symmetrically. □

Proposition 3.5. *The following inference rules are derivable:*

$$\dfrac{}{\vdash P, P^{\perp}}\ \mathit{init} \qquad \dfrac{\vdash \Gamma, B(\nu B)\vec{t}}{\vdash \Gamma, \nu B\vec{t}}\ \nu R$$

Proof. The admissibility of *init* is a standard result — it is also a special case of functoriality. The unfolding νR is derivable from ν, using $B(\nu B)$ as the coinvariant S. The proof of co-invariance $((B(\nu B)\vec{x}) \multimap B(B(\nu B))\vec{x})$ is obtained applying the functoriality of B on the following subderivation:

$$\dfrac{\dfrac{}{\vdash B(\nu B)\vec{x}, \overline{B}(\mu\overline{B})\vec{x}}\ \mathit{init}}{\vdash B(\nu B)\vec{x}, \mu\overline{B}\vec{x}}\ \mu$$

□

Example 3.6. Units can be represented by means of $=$ and $\neq$. Assuming that 2 and 3 are two distinct constants, then we have $2 = 2 \multimap\!\!\circ\ \mathbf{1}$ and $2 = 3 \multimap\!\!\circ\ \mathbf{0}$ (and hence $2 \neq 2 \multimap\!\!\circ\ \perp$ and $2 \neq 3 \multimap\!\!\circ\ \top$). Here, $P \multimap\!\!\circ\ Q$ denotes $\vdash (P \multimap Q)\ \&\ (Q \multimap P)$ and $P \multimap Q$ denotes the formula $P^{\perp} \parr Q$.

Example 3.7. The μ (resp. ν) connective is meant to represent least (resp. greatest) fixed points. For example $\nu(\lambda p.p)$ is provable (take any provable formula as the co-invariant), while its dual $\mu(\lambda p.p)$ is not provable. More precisely: $\mu(\lambda p.p) \multimap\!\!\circ\ \mathbf{0}$ and $\nu(\lambda p.p) \multimap\!\!\circ\ \top$.

Example 3.8. The least fixed point, as expected, entails the greatest. The following is a proof of $\mu B\vec{t} \multimap \nu B\vec{t}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B(\mu B)\vec{x}, \overline{B}(\nu\overline{B})\vec{x}}\ \mathit{init}}{\vdash B(\mu B)\vec{x}, \nu\overline{B}\vec{x}}\ \nu R \qquad \dfrac{}{\vdash \mu B\vec{t}, \nu\overline{B}\vec{t}}\ \mu\nu}{\vdash \nu\overline{B}\vec{t}, \nu B\vec{t}}\ \nu \text{ on } \nu B\vec{t} \text{ with } S := \mu B$$

The greatest fixed point entails the least fixed point when the fixed points are noetherian, *i.e.*, all unfoldings of B and $\overline{B}$ terminate.

3.2 Cut-elimination

In Chapter 2, we have described the cut reductions for fixed points in intuitionistic logic. There is no obstacle to extend the reductions of MALL along the same lines to obtain the reductions for μMALL. The same does not hold for the proof of termination of the cut reductions: the reducibility technique used for the extensions of LJ [MT03b, Tiu04] is not adapted to classical calculi such as μMALL. In our setting, one has to use another technique, such as (variants of) the reducibility candidates used by Girard for the strong normalization of second order linear logic or Matthes [Mat99, Mat98] for his extensions of system F with fixed points. It would be an interesting task to carry out, which could also benefit the intuitionistic systems by establishing the uselessness of the stratification condition used in that case — Tiu conjectured in [Tiu04] that this technical device was not essential, and the work on system F, as well as ours, seems to indicate that monotonicity is enough in these frameworks. However, we leave it to further work and present here an indirect proof.

We prove cut-elimination for μMALL by encoding it into second-order linear logic (LL2), normalizing the encoding and finally translate back the cut-free LL2 derivation to a cut-free μMALL derivation. Although this is indirect, it is still a syntactic proof of cut-elimination. As usual, it yields consistency of μMALL as well as relative soundness and completeness with respect to LL2.

Theorem 3.9. *The logic μMALL enjoys cut-elimination.*

Proof. We first show in Lemma 3.12 how to translate μMALL formulas and proofs into full second-order linear logic derivations, which are then normalized and focused, and finally translated back to cut-free μMALL derivations as shown in Lemma 3.13.

Formally, the normalization has only been established for LL2 without equality, but previous work on equality has shown that it has little role to play in normalization — in general, first-order ingredients do not play a role in the complexity and termination of cut-elimination. Similarly, Andreoli's focusing did not treat first-order structure at all, but it is straightforward to extend it: first-order quantifiers behave just like second-order ones, and the treatment of equality is unsurprising as we shall see in the next chapter. □

To distinguish second-order quantifiers from first-order ones, we shall write them $\forall^2$ and $\exists^2$.

Definition 3.10 (Translation from first-order to second-order)**.** The translation commutes with the connectives of MALL and the first-order ones, as well as the negation. It is defined as follows on the least fixed points:

$$\ulcorner \mu B\vec{t} \urcorner \stackrel{def}{=} \forall^2 S.\ !(\forall \vec{y}.\ \ulcorner B \urcorner S\,\vec{y} \multimap S\,\vec{y}) \multimap S\,\vec{t}$$

And hence we have:

$$\ulcorner \nu B\vec{t} \urcorner \stackrel{def}{=} \ulcorner \mu \overline{B}\vec{t} \urcorner^{\perp} \equiv \exists^2 S.\ !(\forall \vec{y}.\ S\,\vec{y} \multimap \ulcorner B \urcorner S\,\vec{y}) \otimes S\,\vec{t}$$

Here, $\ulcorner B \urcorner$ stands for $\lambda p.\lambda \vec{x}.\ \ulcorner B p \vec{x} \urcorner$, and we define similarly the encoding of a formula abstracted over terms: $\ulcorner S \urcorner$ is $\lambda x.\ \ulcorner S\,x \urcorner$.

That translation is the straightforward linear adaptation of the standard impredicative encoding of least fixed points. Since predicate variables are translated to second-order variables and everything is compositional, we have $\lceil B\rceil\lceil S\,\vec{\rceil} \equiv \lceil BS\,\vec{\rceil}$ and $\lceil S\,\vec{\rceil}\vec{t} \equiv \lceil S\,\vec{t}\rceil$.

Definition 3.11 (Well-restricted second-order instantiations)**.** A valid second-order instantiation is an introduction of $\exists^2$ of one of the following two forms:

$$\frac{\vdash \Theta; \Gamma, P\lceil Q\rceil}{\vdash \Theta; \Gamma, \exists^2 X.PX}\ Q \text{ closed}$$

$$\frac{\vdash \Theta; \Gamma, !(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}) \otimes (S\vec{t})^{\perp}}{\vdash \Theta; \Gamma, \lceil \nu \overline{B}\vec{t}\,\vec{\rceil}}\ !(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}) \in \Theta$$

Lemma 3.12. *If $\vdash \Gamma$ is derivable in μMALL then $\vdash \lceil\Gamma\rceil$ has a proof in LL2. Moreover, every instance the $\exists^2$ rule in that proof is well-restricted.*

Proof. A simple induction allows to translate μMALL derivations rule by rule, which is obvious for all connectives except fixed points. The axiom on fixed points becomes an instance of the identity in LL2. It also works naturally for the ν rule, which is translated as follows:

$$\frac{S\vec{t} \vdash \Gamma \quad BS\vec{x} \vdash S\vec{x}}{\mu B\vec{t} \vdash \Gamma} \quad\rightsquigarrow\quad \frac{\dfrac{\lceil B\rceil\lceil S\,\vec{\rceil}\vec{y} \vdash \lceil S\,\vec{\rceil}\vec{y} \quad \lceil S\,\vec{\rceil}\vec{t} \vdash \lceil\Gamma\rceil}{!(\forall \vec{y}.\ \lceil B\rceil\lceil S\,\vec{\rceil}\vec{y} \multimap \lceil S\,\vec{\rceil}\vec{y}) \multimap \lceil S\,\vec{\rceil}\vec{t} \vdash \lceil\Gamma\rceil}}{\forall S.\ !(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}) \multimap S\vec{t} \vdash \lceil\Gamma\rceil}$$

The validity of the encoding of least fixed points strongly relies on monotonicity. Indeed, the second-order eigenvariable $S\vec{x}$ can simulate the unfoldings of $\mu B\vec{x}$ thanks to the prefixed point hypothesis about it, but if the fixed point got negated by B after an unfolding it could be subject to inductions that S cannot simulate at all. This is formally expressed by the use of functoriality in the LL2 derivation that is cut to translate applications of the μ rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Trivial}{!(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}), \forall S.\ !(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}) \multimap S\vec{x} \vdash S\vec{x}}}{!(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}), \lceil \mu B\rceil\vec{x} \vdash S\vec{x}}}{!(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}), \lceil B\rceil\lceil \mu B\rceil\vec{t} \vdash \lceil B\rceil S\vec{t}}\ functo}{!(\forall \vec{y}.\ \lceil B\rceil S\vec{y} \multimap S\vec{y}), \lceil B\rceil\lceil \mu B\rceil\vec{t} \vdash S\vec{t}}}{\vdash \lceil B\rceil\lceil \mu B\rceil\vec{t} \multimap \lceil \mu B\vec{t}\,\vec{\rceil}}$$

At the end of that derivation we basically have an identity, where a second-order existential is instantiated with the corresponding variable S. It is the only place in the translation where a second-order existential is not instantiated by an $\lceil . \rceil$ encoding. □

Often, encoding a system into another is easy, but decoding can be cumbersome. If one rule of the original system translates to several, and a transformation (*e.g.,* normalization) is applied to the translated derivation, these several rules can end up scattered through the derivation. Here, we remark that focusing does the tedious task of reorganizing those rules for us.

Lemma 3.13. *If* $\vdash \lceil\Gamma\rceil$ *has an LL2 derivation that uses only well-restricted introductions of* $\exists^2$*, then there is a cut-free derivation of* $\vdash \Gamma$ *in μMALL.*

Definition 3.14. Let Θ be a set of formulas. We define the translation $\lfloor . \rfloor_\Theta$ from LL2 to μMALL formulas as follows:

$$\begin{array}{rcll}
\lfloor P \otimes Q \rfloor_\Theta & \equiv & \lfloor P \rfloor_\Theta \otimes \lfloor Q \rfloor_\Theta & \text{and similarly for } \parr, \oplus \text{ and } \& \\
\lfloor \mathbf{1} \rfloor_\Theta & \equiv & \mathbf{1} & \text{and similarly for } \bot, \top, \mathbf{0}, u = v \text{ and } u \neq v \\
\lfloor \forall x.\, Px \rfloor_\Theta & \equiv & \forall x.\, \lfloor Px \rfloor_\Theta & \text{and similarly for } \exists \\
\lfloor S\vec{t} \rfloor_\Theta & \equiv & \mu B\vec{t} & \text{when } S \text{ is a second-order variable} \\
 & & & \quad \text{and } (\forall \vec{y}.\, \lceil B \rceil S\vec{y} \multimap S\vec{y}) \in \Theta
\end{array}$$

And finally:

$$\begin{array}{rcl}
\lfloor \forall^2 S.\, !(\forall \vec{y}.\, \lceil B \rceil S\vec{y} \multimap S\vec{y}) \multimap S\vec{t} \rfloor_\Theta & \equiv & \mu B\vec{t} \\
\lfloor \exists^2 S.\, !(\forall \vec{y}.\, S\vec{y} \multimap \lceil B \rceil S\vec{y}) \otimes S\vec{x} \rfloor_\Theta & \equiv & \nu B\vec{t}
\end{array}$$

We extend this notation to multisets of formulas in the usual way, and omit to specify Θ explicitly as it shall be obvious from the context. We say that a formula is $\lfloor . \rfloor$-encodable when the translation is defined for that formula.

Proof. We first normalize the proof of $\vdash \lceil\Gamma\rceil$ and focus it with all predicate variables being treated asynchronously. Then we proceed by a simple induction on the focused LL2 derivation, precisely establishing that:

1. If there is a proof of $\vdash \Theta : \Gamma \Uparrow \Delta$ where $\Theta = \{(\forall \vec{y}.\lceil B_i \rceil S_i\vec{y} \multimap S_i\vec{y})^\perp \mid i \in \mathcal{I}\}$, and Γ, Δ are multisets of $\lfloor . \rfloor$-encodable formulas, then there is a proof of $\vdash \lfloor \Gamma, \Delta \rfloor$.

2. If there is a proof of $\vdash \Theta : \Gamma \Downarrow P$, with the same condition on Θ and Γ, and with a $\lfloor . \rfloor$-encodable P, then there is a proof of $\vdash \lfloor \Gamma, P \rfloor$.

We conclude by observing that $\lfloor \lceil \Gamma \rceil \rfloor$ is Γ.

The asynchronous cases of the induction are easy. Only the introduction of second-order universal quantifier is not directly mapped to μMALL, but it does not change the $\lfloor . \rfloor$ translation.

When the asynchronous phase ends, a focus is chosen. If it is put on a formula from the linear context Γ, we simply use the second part of our induction hypothesis. But it can also be put on an unfolding hypothesis in Θ, causing its contraction-dereliction. This corresponds to the unfolding of a least fixed point. In that case, the polarity of S forces the derivation to have a convenient form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Theta : \Gamma \Downarrow \lceil B_i \rceil S_i\vec{t}} \quad \dfrac{}{\vdash \Theta : S_i\vec{t} \Downarrow (S_i\vec{t})^\perp}}{\vdash \Theta : \Gamma, S_i\vec{t} \Downarrow \lceil B_i \rceil S_i\vec{t} \otimes (S_i\vec{t})^\perp}}{\vdash \Theta : \Gamma, S_i\vec{t} \Downarrow \exists \vec{y}. \lceil B_i \rceil S_i\vec{y} \otimes (S_i\vec{y})^\perp}}{\vdash \Theta : \Gamma, S_i\vec{t} \Uparrow}$$

By induction hypothesis on the subderivation focused on $\lceil B_i \rceil S_i \vec{t}$, we get a derivation of $\vdash \lfloor \Gamma \rfloor, B_i(\mu B_i)\vec{t}$, from which we build a derivation of the conclusion's decoding, that is $\vdash \lfloor \Gamma \rfloor, \mu B_i \vec{t}$.

We now treat the synchronous case. The propositional and first-order cases translate trivially. The delicate step is when the focus is set on the encoding of a greatest fixed point. It triggers the introduction of a second-order existential, immediately followed by a tensor and an exponential. The crucial remark here is that normalization and focusing do not affect the instantiations of second-order instantiations. Thus, the instance is either of the form $\lceil I \rceil$ for some closed I (case of a (co)induction), or it is the variable associated to some least fixed point (case of an identity). Let us first treat the case of a closed $\lceil I \rceil$:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Theta :\Uparrow \lceil BI\vec{y}\rceil^{\perp}, \lceil I\vec{y}\rceil} \quad \dfrac{\vdots}{\vdash \Theta : \Gamma \Downarrow \lceil I\vec{x}\rceil^{\perp}}}{\vdash \Theta : \Gamma \Downarrow (!\forall \vec{y}.\lceil B\rceil\lceil I\rceil\vec{y} \multimap \lceil I\rceil\vec{y}) \otimes \lceil I\vec{x}\rceil^{\perp}}}{\vdash \Theta : \Gamma \Downarrow \exists^2 S.!(\forall \vec{y}.\lceil B\rceil S\,\vec{y} \multimap S\,\vec{y}) \otimes (S\,\vec{x})^{\perp}}$$

It translates well into:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \lfloor \Gamma \rfloor, (I\vec{x})^{\perp}} \quad \dfrac{\vdots}{\vdash \overline{B}I^{\perp}\vec{y}, I\vec{y}}}{\vdash \lfloor \Gamma \rfloor, \nu\overline{B}\vec{x}}}{\vdash \lfloor \Gamma \rfloor, (\mu B\vec{x})^{\perp}} =$$

Now, if the existential is instantiated by the corresponding variable S, the focusing enforces again the right form:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Theta :\Downarrow !(\forall \vec{y}.\ \lceil B\rceil S_i\vec{y} \multimap I\vec{y})} \quad \dfrac{}{\vdash \Theta : S_i\vec{t} \Downarrow (S_i\vec{t})^{\perp}}}{\vdash \Theta : S\,\vec{t} \Downarrow !(\forall \vec{y}.\ \lceil B\rceil S_i\vec{y} \multimap S\,\vec{y}) \otimes (S_i\vec{t})^{\perp}}}{\vdash \Theta : S\,\vec{t} \Downarrow \exists^2 S.\ !(\forall \vec{y}.\lceil B\rceil S\,\vec{y} \multimap S\,\vec{y}) \otimes (S\,\vec{t})^{\perp}}$$

Since S represents the least fixed point on B, the vertical dots are an instance of the identity and the concluding sequent translates back to $\vdash \mu B\vec{t}, \nu\overline{B}\vec{t}$, which we prove using the $\mu\nu$ rule. □

As shown in the above proof, fixed points can be encoded by means of second-order quantification and exponentials. However, first-order MALL with exponentials and first-order MALL with fixed points are incomparable.

An interesting observation about our technique is that it puts more structure on μMALL derivations than what we claim. Not only do we obtain a cut-free derivation, it also inherits a focused structure from LL2. However, we shall see in the next chapter that the focusing that it yields on fixed points is much weaker than what can be obtained from a closer examination.

We conjecture that a similar technique would work for establishing cut-elimination (and hence consistency) for μLL, *i.e.,* μMALL extended with exponentials. It requires

some extra care because the non-linear context Θ would not be exclusively used to store the formulas stating that second-order variables represent prefixed points, but that should not make an essential difference.

As said in Chapter 2, Girard [Gir92] observed that exponentials and non-monotonic definitions combine to yield inconsistency: for example, the definition $p \equiv p^\perp$ (that is, the fixed point $\mu\lambda p.p^\perp$) does not lead to an inconsistency, whereas the definition $p \equiv ?(p^\perp)$ (that is, $\mu\lambda p.?(p^\perp)$) does. To reproduce the latter inconsistency in μMALL, one needs to be able to unfold the expression $\nu\lambda p.!(p^\perp)$. But this is not implied by Proposition 3.5 since its body is not monotonic. Thus, even in the presence of exponentials, we currently do not have any example of non-monotonic definition that invalidates the consistency of μMALL.

3.3 Classification of connectives

It is common to classify inference rules between invertible and non-invertible ones. In linear logic, we can use the refined notion of positive and negative connectives (*cf.* Definition 1.9). It is more interesting, as we learn something about positive connectives, while non-invertibility is only an absence of information. Even if it does not seem to be related to proof-search, positivity turns out to play an important role in the understanding and design of focused system [LM07a, Lau02, LQdF05, DJS93, DJS95].

Since μMALL does not have exponentials, it is not possible to talk about positivity of its formulas and connectives. Hence, we are going to take a backwards approach: we shall forget about the definition of positivity and redefine it syntactically, then prove that negative formulas have a property close to the original negativity. Finally, we consider the logic extended with exponentials and establish that the original notion of positivity holds for our syntactically defined positives.

3.3.1 Polarities in μMALL

Definition 3.15. We classify as *negative* the following connectives: $\parr$, $\perp$, $\&$, $\top$, $\forall$, $\neq$, ν. Their duals are called *positive*. A formula is said to be negative (resp. positive) when all of its connectives are negative (resp. positive). Finally, an operator $\lambda p\lambda\vec{x}.B p\vec{x}$ is said to be negative (resp. positive) when the formula $Bp\vec{x}$ is negative (resp. positive).

Notice, for example, that $\lambda p\lambda\vec{x}.p\vec{x}$ is both positive and negative. But $\mu p.p$ is only positive while $\nu p.p$ is only negative.

Proposition 3.16. *The following structural rules are admissible provided that B is negative:*

$$\frac{\vdash \Gamma, \nu B\vec{t}, \nu B\vec{t}}{\vdash \Gamma, \nu B\vec{t}}\ \nu C \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \nu B\vec{t}}\ \nu W$$

Hence, the following structural rules hold for any negative formula P:

$$\frac{\vdash \Gamma, P, P}{\vdash \Gamma, P}\ C \qquad \frac{\vdash \Gamma}{\vdash \Gamma, P}\ W$$

Proof. The standard argument for obtaining the general rules (outlined in Chapter 1, Section 1.4) works except for the new case of the greatest fixed point. Hence, we only show the proof of admissibility for the greatest fixed point rules, especially since that proof also contains the ingredients of the general one.

We first prove the admissibility of νW. It is obtained by co-induction, choosing $\bot$ as the co-invariant. We obtain the co-invariance proof $(\vdash B(\lambda\vec{x}.\bot)\vec{x}, \mathbf{1})$ from a more general result: for any collection of negative formulas $(P_i)_i$, there is a derivation of $\vdash (P_i)_i, \mathbf{1}$. This is done by induction on the total size of $(P_i)_i$, counting one for each connective, unit and predicate variable but ignoring terms. The proof is trivial if the collection is empty. Otherwise, if P_0 is a disequality we conclude by induction with one less formula, and the size of the others unaffected by the unification; if it's $\top$ our proof is done; if it's $\bot$ then P_0 disappears and we conclude by induction hypothesis. The $\parr$ case is done by induction hypothesis, the resulting collection has one more formula but is smaller; the & makes use of two instances of the induction hypothesis. Finally, the ν case is done by applying the ν rule with $\bot$ as the invariant:

$$\dfrac{\dfrac{\vdash (P_i)_i, \mathbf{1}}{\vdash \bot, (P_i)_i, \mathbf{1}} \quad \vdash B(\lambda\vec{x}.\bot)\vec{x}, \mathbf{1}}{\vdash \nu B\vec{t}, (P_i)_i, \mathbf{1}}$$

The two subderivations are obtained by induction. For the second one there is only one formula, namely $B(\lambda\vec{x}.\bot)\vec{x}$, which is indeed negative (by monotonicity of B) and smaller than νB.

Contraction (νC) is also an instance of the ν rule, obtained by choosing the co-invariant $(\lambda\vec{x}.\ \nu B\vec{x} \parr \nu B\vec{x})$ — in the following, we write it $(\nu B \parr \nu B)$ for short. As before, we shall build, by induction on B, a derivation of the co-invariance of that formula. Unfortunately, it is not possible to obtain directly a cut-free one. The proof of co-invariance $(\nu B\vec{x} \parr \nu B\vec{x} \vdash B(\nu B \parr \nu B)\vec{x})$ follows from that of $B\nu B\vec{x} \parr B\nu B\vec{x} \vdash B(\nu B \parr \nu B)\vec{x}$ — a first cut is used for replacing the fixed points by their unfoldings, which is equivalent. We generalize over that form of sequent and derive in μMALL, for any negative n-ary operator A:

$$A(\nu B_1)\dots(\nu B_n) \parr A(\nu B_1)\dots(\nu B_n) \vdash A(\nu B_1 \parr \nu B_1)\dots(\nu B_n \parr \nu B_n)$$

We prove this by induction on A:

- It is trivial if A is a disequality, $\top$ or $\bot$.
- A is a projection $\lambda p_1 \dots \lambda p_n.p_i\vec{t}$: we have to derive $\nu B_i\vec{t} \parr \nu B_i\vec{t} \vdash \nu B_i\vec{t} \parr \nu B_i\vec{t}$, which is an instance of *init*.
- A is $A_1 \parr A_2$. In general, $(P_1 \parr P_2) \parr (P_1 \parr P_2)$ is provably equivalent to $(P_1 \parr P_1) \parr (P_2 \parr P_2)$. We use this for $P_i := A_i(\nu B_i)_i$, cutting it to replace the left hand-side of our sequent by $(A_1(\nu B_i)_i \parr A_1(\nu B_i)_i) \parr (A_2(\nu B_i)_i \parr A_2(\nu B_i)_i)$. We can then introduce the right hand-side $\parr$, and split the left one, to finally conclude by induction hypothesis on A_1 and A_2.

- A is $A_1 \ \& \ A_2$. We introduce the additive conjunction on the right, and have to derive two similar premises:

$$(A_1 \ \& \ A_2)(\nu B_i)_i \parr (A_1 \ \& \ A_2)(\nu B_i)_i \vdash A_1(\nu B_i \parr \nu B_i)_i$$

  To conclude by induction hypothesis, we have to choose the right projections for the left hand-side &. Since the & is under the $\parr$, we have to use a cut — one can derive in general that $(P_1 \ \& \ P_2) \parr (P_1 \ \& \ P_2) \vdash P_1 \parr P_1$.

- A is $\lambda p1 \dots \lambda p_n.\ \forall x.\ A'(p_i)_i x$. The same scheme applies: we introduce the universal variable on the right, and instantiate the two quantifiers on the left, under the $\parr$ thanks to a cut.

- A is $\lambda p1 \dots \lambda p_n.\ \nu(A' p_1 \dots p_n)\vec{t}$. Let B_{n+1} be $A'(\nu B_i)_{i \le n}$. We build our derivation as follows:

$$\dfrac{\dfrac{Id \qquad \dfrac{A'(\nu B_i)_i(\nu B_{n+1})\vec{x} \parr A'(\nu B_i)_i(\nu B_{n+1})\vec{x} \vdash A'(\nu B_i \parr \nu B_i)_i(\nu B_{n+1} \parr \nu B_{n+1})\vec{x} \quad Trivial}{\nu B_{n+1}\vec{x} \parr \nu B_{n+1}\vec{x} \vdash A'(\nu B_i \parr \nu B_i)_i(\nu B_{n+1} \parr \nu B_{n+1})\vec{x}}\ cut}{\nu B_{n+1}\vec{t} \parr \nu B_{n+1}\vec{t} \vdash \nu(A'(\nu B_i \parr \nu B_i)_i)\vec{t}}\ \nu}{\nu(A'(\nu B_i)_i)\vec{t} \parr \nu(A'(\nu B_i)_i)\vec{t} \vdash \nu(A'(\nu B_i \parr \nu B_i)_i)\vec{t}}$$

  The *Trivial* derivation that is cut in establishes $\nu B_{n+1}\vec{x} \parr \nu B_{n+1}\vec{x} \vdash A'(\nu B_i)_i(\nu B_{n+1})\vec{x} \parr A'(\nu B_i)_i(\nu B_{n+1})\vec{x}$ — there is even an equivalence between the two because all we did is unfold νB_{n+1}. We finally complete the derivation above by induction hypothesis, with B_{n+1} added to the family $(B_i)_i$ and the smaller operator expression A'.

□

The previous property is interesting in that it is established in an exponential-free framework. However, in that setting, while negatives are characterized by admissible rules, there is nothing to say about positives. To fix that, let us move temporarily to μLL, that is μMALL plus exponentials.

3.3.2 Polarities in μLL

Instead of only establishing the usual characterization of positivity for the positive formulas of μMALL, we observe that it can be obtained for a larger class of formulas. Doing so, we essentially extend the syntactic positivity criterion to μLL by observing how exponentials and fixed points interact. We expect that such observations would be useful to study μLJ through the looking glass of μLL.

Definition 3.17. Positive formulas are given by the following grammar:

$$\begin{array}{rcl} \mathcal{P} & ::= & \mathbf{1} \mid \mathcal{P} \otimes \mathcal{P} \mid \mathbf{0} \mid \mathcal{P} \oplus \mathcal{P} \mid \exists x.\ \mathcal{P} \mid u = v \\ & \mid & !Q \text{ for any formula Q} \\ & \mid & \mu B\vec{t} \text{ when } B \text{ is } \textit{weakly positive}, \textit{ i.e., } B\mathbf{1}\vec{t} \in \mathcal{P} \\ & \mid & \nu B\vec{t} \text{ when } B \text{ is } \textit{strongly positive}, \textit{ i.e., } B\bot\vec{t} \in \mathcal{P} \end{array}$$

Negative formulas are those whose dual is positive.

Proposition 3.18. *Any positive formula P is provably equivalent to $!P$. Said otherwise, any negative formula Q is provably equivalent to $?Q$.*

On the first line, we find the positive formulas of μMALL. It is well-known that $!Q$ is positive independently of Q, since $!Q \multimap !!Q$. That creates the possibility to have *strongly positive* bodies and positive greatest fixed points. A simple examination of the definition shows that if $B\bot$ is positive, then any occurrence of p in $(\lambda p.\ Bp)$ must be under a !. Hence $B(\nu B)$ is also positive, and so is νB because they are equivalent.

Proof. We prove the following generalization, by induction on the weakly positive operator expression B: for any (possibly empty) collection of formulas $(P_i)_i$, it is provable that $B(!P_i)_i \vdash !B(!P_i)_i$. This entails our proposition, since a positive formula can be seen as a weakly positive operator expression with a vacuous abstraction.

Most cases of the induction are standard: the connectives $\otimes, \oplus, \exists$ preserve positivity, and $\mathbf{1}, \mathbf{0}$ and equality are positive — in the sense of being equivalent to its own !. The case of the least fixed point is as follows, where P denotes $\mu B'(!P_i)_i$, and Θ is obtained by induction hypothesis:

$$\dfrac{Id \qquad \dfrac{\dfrac{\dfrac{\Theta'}{B'(!P_i)_i!P \vdash !B'(!P_i)_i!P} \quad \dfrac{\dfrac{\dfrac{\dfrac{\overline{P\vec{z} \vdash P\vec{z}}}{!P\vec{z} \vdash P\vec{z}}}{B'(!P_i)_i!P\vec{x} \vdash B'(!P_i)_iP\vec{x}}\ functo}{B'(!P_i)_i!P\vec{x} \vdash \mu B'(!P_i)\vec{x}}}{!B'(!P_i)_i!P\vec{x} \vdash !\mu B'(!P_i)\vec{x}}}{B'(!P_i)_i(\lambda\vec{y}.\ !\mu B'(!P_i)_i\vec{y})\vec{x} \vdash !\mu B'(!P_i)\vec{x}}\ cut}{\mu B'(!P_i)_i\vec{t} \vdash !\mu B'(!P_i)_i\vec{t}}\ \nu$$

A simple examination of the definition of positivity shows that in a strongly positive operator $(\lambda p.\ B'p)$, all occurrences of p are under a !, although not necessarily immediately. Hence, a given instance $B'P$ can be written as $B''(!Q_i)_i$. This allows us to treat the case of a greatest fixed point over a strongly positive body by simply unfolding the fixed points on both sides (under the ! on the right hand-side). Here, Θ is again obtained by induction hypothesis, by observing that the sequent that it has to prove can be written as $B''(!P_i)_i(!Q_j)_j\vec{t} \vdash !B''(!P_i)_i(!Q_j)_j\vec{t}$:

$$\dfrac{\dfrac{\Theta}{B'(!P_i)_i(\nu B'(!P_i)_i)\vec{t} \vdash !B'(!P_i)_i(\nu B'(!P_i)_i)\vec{t}}}{\nu B'(!P_i)_i\vec{t} \vdash !B'(!P_i)_i(\nu B'(!P_i)_i)\vec{t}} \quad \dfrac{\dfrac{\overline{B'(!P_i)_i(\nu B'(!P_i)_i)\vec{t} \vdash B'(!P_i)(\nu B'(!P_i)_i)\vec{t}}\ init}{B'(!P_i)_i(\nu B'(!P_i)_i)\vec{t} \vdash \nu B'(!P_i)_i\vec{t}}\ \nu R}{!B'(!P_i)_i(\nu B'(!P_i)_i)\vec{t} \vdash !\nu B'(!P_i)_i\vec{t}}$$
$$\dfrac{}{\nu B'(!P_i)_i\vec{t} \vdash !\nu B'(!P_i)_i\vec{t}}\ cut$$

□

3.4 Examples

We shall now give a few theorems in μMALL. Although we do not give their derivations here but only a quick description of how to obtain it, we stress that all of these

examples are proved naturally. We also invite the reader to check that the μ-focusing system presented in Chapter 4 is an useful guide when deriving these example, leaving only the important choices. The reader will also note that although μMALL is linear, these derivations are intuitive and their structure resemble that of proofs in intuitionistic logic.

We first define a few least fixed points expressing basic properties of natural numbers. We assume two constants 0 and s of respective types n and $n \to n$. Note that all these definitions are positive.

$$\begin{array}{lcl}
nat & \stackrel{def}{=} & \mu(\lambda nat \lambda x.\ x = 0 \oplus \exists y.\ x = s\ y \otimes nat\ y) \\
even & \stackrel{def}{=} & \mu(\lambda even \lambda x.\ x = 0 \oplus \exists y.\ x = s\ (s\ y) \otimes even\ y) \\
plus & \stackrel{def}{=} & \mu(\lambda plus \lambda a \lambda b \lambda c.\ a = 0 \otimes b = c \\
 & & \qquad \oplus \exists a' \exists c'.a = s\ a' \otimes c = s\ c' \otimes plus\ a'\ b\ c') \\
leq & \stackrel{def}{=} & \mu(\lambda leq \lambda x \lambda y.\ x = y \oplus \exists y'.\ y = s\ y' \otimes leq\ x\ y') \\
half & \stackrel{def}{=} & \mu(\lambda half \lambda x \lambda h.\ (x = 0 \oplus x = s\ 0) \otimes h = 0 \\
 & & \qquad \oplus \exists x' \exists h'.\ x = s\ (s\ x') \otimes h = s\ h' \otimes half\ x'\ h') \\
ack & \stackrel{def}{=} & \mu(\lambda ack \lambda m \lambda n \lambda a.\ m = 0 \otimes a = s\ n \\
 & & \qquad \oplus (\exists p.\ m = s\ p \otimes n = 0 \otimes ack\ p\ (s\ 0)\ a) \\
 & & \qquad \oplus (\exists p \exists q \exists b.\ m = s\ p \otimes n = s\ q \otimes ack\ m\ q\ b \otimes ack\ p\ b\ a))
\end{array}$$

The following statements are theorems in μMALL. The main insights required for proving these theorems involve deciding which fixed point expression should be introduced by induction: the proper invariant is not the difficult choice here since the context itself is adequate in these cases.

$$\begin{array}{l}
\vdash \forall x.\ nat\ x \multimap even\ x \oplus even\ (s\ x) \\
\vdash \forall x.\ nat\ x \multimap \forall y \exists z.\ plus\ x\ y\ z \\
\vdash \forall x.\ nat\ x \multimap plus\ x\ 0\ x \\
\vdash \forall x.\ nat\ x \multimap \forall y.\ nat\ y \multimap \forall z.\ plus\ x\ y\ z \multimap nat\ z
\end{array}$$

In the last theorem, the assumption $(nat\ x)$ is not needed and can be weakened, thanks to Proposition 6.11. In order to prove $(\forall x.\ nat\ x \multimap \exists h.\ half\ x\ h)$ the context does not provide an invariant that is strong enough. A typical solution is to use complete induction, *i.e.,* use the strengthened invariant $(\lambda x.\ nat\ x \otimes \forall y.\ leq\ y\ x \multimap \exists h.\ half\ y\ h)$.

A typical example of co-induction involves the simulation relation. Assume that $step : state \to label \to state \to o$ is an inductively defined relation encoding a labeled transition system. Simulation can be defined using the definition

$$sim \stackrel{def}{=} \nu(\lambda sim \lambda p \lambda q.\ \forall a \forall p'.\ step\ p\ a\ p' \multimap \exists q'.\ step\ q\ a\ q' \otimes sim\ p'\ q').$$

Reflexivity of simulation $(\forall p.\ sim\ p\ p)$ is proved easily by co-induction with the co-invariant $(\lambda p \lambda q.\ p = q)$. Instances of $step$ are not subject to induction but are treated "as atoms". Proving transitivity, that is,

$$\forall p \forall q \forall r.\ sim\ p\ q \multimap sim\ q\ r \multimap sim\ p\ r$$

is done by co-induction on $(sim\ p\ r)$ with the co-invariant $(\lambda p \lambda r.\ \exists q.\ sim\ p\ q \otimes sim\ q\ r)$. The focus is first put on $(sim\ p\ q)^\perp$, then on $(sim\ q\ r)^\perp$. The fixed points $(sim\ p'\ q')$ and $(sim\ q'\ r')$ appearing later in the proof are treated "as atoms", as are all instances of $step$. Notice that these two examples are also cases where the context gives a coinvariant.

Except for the totality of *half*, all these theorems seem simple to prove using a limited number of heuristics. For example, one could first try to treat fixed points "as atoms", an approach that would likely fail quickly if inappropriate. Second, depending on the "rigid" structure of the arguments to a fixed point expression, one might choose to either unfold the fixed point or attempt to use the surrounding context to generate an invariant. Doing this without a good focusing system would be very inefficient, but we shall see in Chapter 7 that a simple strategy can go a long way when applied on top of the focused system developed in Chapter 4.

3.5 Expressiveness

The study of μMALL raises the question of how far one can go without the exponentials, getting the infinite behavior from the meaning of fixed points instead of modalities. This question has interested people for long, for the particular case of arithmetic: having natural numbers and induction, is it possible to represent the same functions with and without contractions? This question has been asked in categorical terms by Burroni [Bur86]. Taking a computational approach, the authors of [AFFM06] show that Gödel's System T can be restricted to follow a linear discipline without loosing in expressiveness. These studies start with the observation that *nat* can be contracted and weakened by means of induction, which is a particular case of the property of negative formulas (Proposition 6.11). We outline here how this observation is exploited to encode any primitive recursive function in μMALL.

The encoding follows the proof-as-program approach. We assume a system of cut reductions extending that of MALL with reductions for fixed points as outlined in Chapter 2. We define the type of natural numbers as $N := \mu X.\mathbf{1} \oplus X$, and the representation of a numbers in μMALL as follows:

$$\Pi_0 = \dfrac{\dfrac{\dfrac{}{\vdash \mathbf{1}}}{\vdash \mathbf{1} \oplus N}}{\vdash N} \qquad \Pi_{S(n)} = \dfrac{\dfrac{\dfrac{\Pi_n}{\vdash N}}{\vdash \mathbf{1} \oplus N}}{\vdash N}$$

A primitive recursive function f of arity k will be represented by a derivation Π_f of $(N)^k \vdash N$, such that for any natural numbers $n_1, \dots, n_k$, the derivation $cut(\Pi; \Pi_{n_1}, \dots, \Pi_{n_k})$ reduces to the representation $\Pi_{f(n_1,\dots,n_k)}$ and no other derivation. To do so, we encode in μMALL the elementary primitive recursive functions (zero, successor and projectors) and simulate the two means to combine them (composition and primitive recursion).

The successor is represented by:

$$\dfrac{\dfrac{\overline{N \vdash N}}{N \vdash \mathbf{1} \oplus N}}{N \vdash N}$$

When cut against a natural number, the auxiliary reductions push the cut above the rules μ and $\oplus$ and replace the axiom by the natural number, effectively forming the representation of its successor.

The projector p_n^i, defined by $p_n^i(x_1, \ldots, x_n) = x_i$, is obtained by weakening the discarded arguments, which is admissible for N:

$$\dfrac{\overline{x_i : N \vdash N}}{\overline{\overline{x_1 : N, \ldots, x_i : N, \ldots, x_n : N \vdash N}}}$$

Notice that the reduction of a projector is not immediate at all, as each discarded natural number is iterated over for building a proof of $\mathbf{1}$ that is finally cut against the main derivation and discarded.

Given an n-ary primitive recursive function f, and n primitive recursive functions $(g_i)_{1 \leq i \leq n}$ of arity m, their *composition* h is also a primitive recursive function:

$$h(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$$

We represent it by contracting n times each of the m inputs and distributing them to the functions g over n successive cuts. We show the derivation for $n = 2$:

$$\dfrac{\dfrac{\Pi_{g_1}}{x_1 : N, \ldots, x_m : N \vdash N} \quad \dfrac{\dfrac{\Pi_{g_2}}{x_1 : N, \ldots, x_m : N \vdash N} \quad \dfrac{\Pi_f}{r_1 : N, r_2 : N \vdash N}}{r_1 : N, x_1 : N, \ldots, x_m : N \vdash N}\ cut}{x_1 : N, \ldots, x_m : N \vdash N}$$

Again, the reduction will be far from efficient, as every natural number will be copied by induction. To be formal, we should check that the contraction given by Proposition 6.11 indeed corresponds to a copy. An inspection of its proof should convince the reader.

Finally, the *primitive recursion* scheme allows to form a function h of arity $n + 1$ from the functions f of arity n and g of arity $n + 2$, defined by:

$$\begin{aligned} h(0, \vec{y}) &= f(\vec{y}) \\ h(S(k), \vec{y}) &= g(k, h(k, \vec{y}), \vec{y}) \end{aligned}$$

We simulate this in μMALL using induction. The computational content of an induction is the iteration of the proof of (co)invariance. This does not match exactly with the recursion scheme, which also carries along the extra parameters $\vec{y}$ and the current rank k. We obtain the same behavior by using an invariant $(N^n \multimap N) \otimes N$: the first component represents the function h partially applied to k, the second component being

k. The derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{N \vdash N}}{(N^n \multimap N), N^n \vdash N}}{(N^n \multimap N), N, N^n \vdash N}}{(N^n \multimap N) \otimes N, N^n \vdash N} \quad \dfrac{\dfrac{\Pi_f \otimes \Pi_0}{\vdash (N^n \multimap N) \otimes N} \quad \dfrac{\dfrac{\dfrac{\Pi_g}{h_k \vec{y} : N, k : N, \vec{y} : N^n \vdash N}}{h_k : (N^n \multimap N), k : N, \vec{y} : N^n \vdash N} \quad \dfrac{\Pi_{succ}}{k : N \vdash Sk : N}}{h_k : (N^n \multimap N), k : N \vdash (N^n \multimap N) \otimes N}}{(N^n \multimap N) \otimes N \vdash (N^n \multimap N) \otimes N}}{\mathbf{1} \oplus (N^n \multimap N) \otimes N \vdash (N^n \multimap N) \otimes N}}{x : N, \vec{y} : N^n \vdash N}$$

When cut against a natural number n, the reduction will iterate n times the invariance premise corresponding to the successor case, starting with the invariance premise corresponding to the base case. The result, of type $(N^n \multimap N) \otimes N$, is then cut against the main premise of the induction, which discards the second component and applies the first one to $\vec{y}$. One can read from the derivation that the iterative process starts with $(f, 0)$ and at each step transforms (h_k, k) into $((\lambda y.\ g(k, h_k, \vec{y})), S(k))$, which corresponds to the primitive recursion scheme.

Clearly, the encoding of primitive recursive functions exploits only a small portion of μMALL. The natural next question is whether μMALL can capture Gödel's System T. In [AFFM06], a linear restriction of System T is designed and proved to be equivalent to the original system. However, their computational notion of linearity, allowing open invariants that only need to be closed at reduction, does not correspond exactly to linear logic. While we can reproduce in μMALL some other of their observations, it is impossible to obtain the copy for arrow types in our system, *i.e.,* $(P \multimap Q) \vdash (P \multimap Q) \otimes (P \multimap Q)$. That contraction lacks, for example, when trying to represent the Ackermann function.

3.6 Conclusion

The logic μMALL is a natural system to consider for the study of least and greatest fixed points. It is very symmetric, and all of its components are canonical. Moreover, while it is based on linear logic, it has a surprising expressiveness and lends itself to traditional reasoning on (co)inductive specifications: arithmetic, algorithms, etc. The rest of this thesis builds on top of that system, exploiting its qualities.

This chapter leaves several questions open. One of the most important future work would be to obtain a proof of cut-elimination that establishes the termination of cut reductions directly on μMALL: this would certainly give insights on the dynamics and complexity of our system. Although various observations rely on monotonicity (unfolding of greatest fixed points, functoriality, the encoding on which our cut-elimination proof is based, but also the definition of cut reductions as in Chapter 2) we do not have any counter-example to consistency in the presence of fixed points on non-monotonic bodies, even when adding exponentials. This question may not have a high practical impact as it is unusual, and very inconvenient, to work with non-monotonic operators. There are also several open questions that are related to the second-order features of (co)induction. In our proof of cut-elimination, and more precisely Lemma 3.13 that decodes an LL2 derivation into a μMALL, one can wonder whether the proviso on

the second-order instantiations can be removed. More likely, and more interestingly, a similar question would arise when relating intuitionistic logic and μLL: if a μLL proof uses an invariant that is not the encoding of an intuitionistic invariant, is there still a way to obtain an intuitionistic proof? This is close to the question of the precise expressiveness of μMALL. Attacking these problems is hard, and might require some new approaches.

Chapter 4

Focusing μMALL

Arguably, the most important property of a logic is its consistency. In sequent calculus, consistency is obtained from cut-elimination, which requires a symmetry between one connective and its dual, or in other word between construction and elimination, conclusion and hypothesis. The notions of polarity and focusing are more recent in proof-theory but their growing importance put them on par with cut-elimination. Focusing organizes proofs in stripes of invertible and non-invertible rules, removing irrelevant interleavings and inducing a reading of the logic based on macro-connectives aggregating stripes of usual connectives. Focusing is useful to justify game theoretic semantics [MS06] and have been central to the design of Ludics [Gir01].

From the viewpoint of proof search, cut-elimination is an essential result as it usually guarantees that no invention is needed. Focusing plays the essential role of reducing the space of the search for a cut-free proof, by identifying situations when backtracking is unnecessary. In logic programming, it plays the more demanding role of correlating the declarative meaning of a program with its operational meaning, given by proof-search. Various computational systems have employed different focusing theorems: much of Prolog's design and implementations can be justified by the completeness of SLD-resolution [AvE82]; uniform proofs (goal-directed proofs) in intuitionistic and intuitionistic linear logics have been used to justify λProlog [MNPS91] and Lolli [HM94]; the classical linear logic programming languages LO [AP91] and Forum [Mil96] have used directly Andreoli's general focusing result [And92] for linear logic.

Inductive proof-search, *e.g.,* in μMALL, is very problematic because cut-free derivations are not analytic: there is no way to restrict the higher-order (relation) variables used in the (co)induction rules. Many systems use various heuristics to restrict the search space, but these solutions lack a proof theoretical justification. In that setting, focusing becomes especially interesting: it yields a restriction of the search space while preserving completeness. Obviously, there is no hope that it will make proof search decidable, but it is an appreciable leap forward, pushing further the limit where proof-theory and completeness leave place to heuristics.

A good focused proof system for μMALL is not a simple consequence of the translation of fixed points into LL2 that is used in the proof of Theorem 3.9: applying linear logic focusing to the result of that translation leads to a poorly structured system that

is not consistent with our classification of connectives as positives and negatives. Let us recall the encoding of $\mu B\vec{t}$:

$$\forall S.\ !(\forall \vec{x}.\ BS\,\vec{x} \multimap S\,\vec{x}) \multimap S\,\vec{t}$$

Superficially, this is asynchronous. Once the asynchronous layer has been processed, one obtains unfoldings by focusing on the pre-fixed-point hypothesis. Through that encoding, one would thus obtain a system where several unfoldings necessarily require several phase alternations. This is not satisfying: the game-based reading of focusing identifies fully synchronous (positive) formulas with data types, which should be built in one step of the player, *i.e.,* in one synchronous phase. In μMALL, least fixed points over fully synchronous bodies should be seen as data types. That intuition, visible in previous examples, is also justified by the classification of connectives (Definition 3.15), and shall be accounted for in our focused system.

In the design of our focusing calculus, and in the proof of its completeness, it is important to keep in mind that invertibility is not asynchrony. Invertibility is the possibility of applying a rule without ever loosing provability, hence both μ and ν are invertible — with an appropriate coinvariant for ν, typically the unfolding invariant. This does not capture the essential aspect of fixed points, that is their infinite behavior. A system requiring that the μ rule is applied whenever possible would not be complete, notably failing on $\vdash \top \otimes \mathbf{1}, \mu X.\ X$ or $\vdash nat\ x \multimap nat\ x$. The best proof of the latter sequent consists in an axiom. Invertibility of μ does give us a proof starting with an unfolding: it is the η-expansion of that axiom. This shows very well that invertibility (at the cost of expansions) is not the thing to consider. Rather, focusing is about permutations of rules in an existing derivation: asynchronous rules commute with any other, synchronous rules commute with other synchronous rules.

The rest of this chapter is organized as follows. We first design the μ-focused system in Section 4.1, treating μ synchronously, which is satisfying for several reasons starting with its positive nature. In Section 4.2, we apply this system to an interesting fragment of μLJ. We show in Section 4.3 that it is also possible to consider a focused system for μMALL where ν is treated synchronously. Finally, we consider in Section 4.4 how to extend our results beyond μMALL, considering μLL and μLJ.

4.1 A complete μ-focused calculus

We present the proof system in Figure 4.1 as a good candidate for a focused proof system for μMALL. We use explicit annotations of the sequents in the style of Andreoli. In the synchronous phase, sequents have the form $\vdash \Gamma \Downarrow P$. In the asynchronous phase, they have the form $\vdash \Gamma \Uparrow \Delta$ where Γ and Δ are both multisets of formulas. In both sequents, Γ is a multiset of synchronous formulas and ν-expressions. The convention on Δ is a slight departure from Andreoli's original proof system where Δ is a list — which can be used to provide a fixed but arbitrary ordering of the asynchronous phase.

The rules for equality are not surprising. The main novelty here is the treatment of fixed points. Each of the fixed point connectives has two rules in the focused system: one treats it "as an atom" and the other one as an expression with "internal structure."

Asynchronous phase

$$\frac{\vdash \Gamma \Uparrow P, Q, \Delta}{\vdash \Gamma \Uparrow P \parr Q, \Delta} \qquad \frac{\vdash \Gamma \Uparrow P, \Delta \quad \vdash \Gamma \Uparrow Q, \Delta}{\vdash \Gamma \Uparrow P \,\&\, Q, \Delta}$$

$$\frac{\vdash \Gamma \Uparrow \Delta}{\vdash \Gamma \Uparrow \bot, \Delta} \qquad \frac{}{\vdash \Gamma \Uparrow \top, \Delta} \qquad \frac{\{\vdash \Gamma\theta \Uparrow \Delta\theta : \theta \in csu(s \doteq t)\}}{\vdash \Gamma \Uparrow s \neq t, \Delta}$$

$$\frac{\vdash \Gamma \Uparrow Pc, \Delta}{\vdash \Gamma \Uparrow \forall x.Px, \Delta}\ c \text{ new}$$

$$\frac{\vdash \Gamma \Uparrow S\,\vec{t}, \Delta \quad \vdash \Uparrow BS\,\vec{x}, S\,\vec{x}^{\perp}}{\vdash \Gamma \Uparrow \nu B\vec{t}, \Delta}\ \vec{x} \text{ new} \qquad \frac{\vdash \Gamma, \nu B\vec{t} \Uparrow \Delta}{\vdash \Gamma \Uparrow \nu B\vec{t}, \Delta}$$

Synchronous phase

$$\frac{\vdash \Gamma \Downarrow P \quad \vdash \Gamma' \Downarrow Q}{\vdash \Gamma, \Gamma' \Downarrow P \otimes Q} \qquad \frac{\vdash \Gamma \Downarrow P_i}{\vdash \Gamma \Downarrow P_0 \oplus P_1}$$

$$\frac{}{\vdash \Downarrow \mathbf{1}} \qquad \frac{}{\vdash \Downarrow t = t}$$

$$\frac{\vdash \Gamma \Downarrow Pt}{\vdash \Gamma \Downarrow \exists x.Px}$$

$$\frac{\vdash \Gamma \Downarrow B(\mu B)\vec{t}}{\vdash \Gamma \Downarrow \mu B\vec{t}} \qquad \frac{}{\vdash \nu\overline{B}\vec{t} \Downarrow \mu B\vec{t}}$$

Switching (where P is synchronous, Q asynchronous)

$$\frac{\vdash \Gamma, P \Uparrow \Delta}{\vdash \Gamma \Uparrow P, \Delta} \qquad \frac{\vdash \Gamma \Downarrow P}{\vdash \Gamma, P \Uparrow} \qquad \frac{\vdash \Gamma \Uparrow Q}{\vdash \Gamma \Downarrow Q}$$

Figure 4.1: The μ-focused proof-system for μMALL

In accordance with Definition 3.15, μ is treated during the synchronous phase and ν during the asynchronous phase.

Roughly, what the focused system implies is that if a proof involving a ν-expression proceeds by co-induction on it, then this co-induction can be done at the beginning; otherwise that formula can be ignored in the whole derivation, except for the $\mu\nu$ rule. The latter case is expressed by the rule which moves the greatest fixed point to the left zone, an operation that we call *freezing*. Focusing on a μ-expression yields two choices: unfolding or applying the initial rule for fixed points. If the body is fully synchronous, the focus will never be lost. For example, if *nat* is the (fully synchronous) expression $\mu(\lambda nat.\lambda x.\ x = 0 \oplus \exists y.x = s\ y \otimes nat\ y)$, then focusing puts a lot of structure on a proof of $\vdash \Gamma \Downarrow nat\ t$: either t is a ground term representing a natural number and Γ is empty, or $t = s^n x$ for some $n \geq 0$ and Γ is $\{(nat\ x)^\perp\}$.

We shall now proceed with the proof of completeness. The main lines of this proof follow [MS07, Sau08] which has two advantages. First, it identifies focused proofs as standard ones with extra structure. The annotated sequents of the focused system are just a concise way of expressing this extra structure. The practical outcome of this is that the proof of completeness is a process of proof transformation working on standard derivations. This proof transformation proceeds by iterating two lemmas which themselves perform rule permutations. The first lemma expresses the fact that asynchronous rules can always be applied first, while the second lemma expresses that synchronous rules can be hereditarily applied once the focus has been chosen. The second benefit of that technique is the clean presentation of the focalization graph for analyzing dependencies in a proof and showing that there is always at least one possible focus.

4.1.1 Balanced derivations

In order to ensure that the focalization process terminates, we have to guarantee that the permutation lemmas do not deepen derivations. Unfortunately, the $\top$ rule is known to cause problems in that respect. For example, here is a possible focalization of an already focused proof:

$$\dfrac{}{\vdash \mathbf{1} \parr \mathbf{1}, \top}\ \top \quad \xrightarrow{foc} \quad \dfrac{\dfrac{}{\vdash \mathbf{1}, \mathbf{1}, \top}\ \top}{\vdash \mathbf{1} \parr \mathbf{1}, \top}\ \parr$$

It is clearly not a major issue, which can be fixed in various ways. A simple solution is to postpone applications of $\top$ after all possible asynchronous rules, before running the focalization process. This is easily obtained in standard MALL, as the eager application of the other asynchronous rules $\parr$, $\&$, $\perp$ and $\forall$ terminates and leaves $\top$ available.

A slight novelty here is that trivial disequalities such as $1 \neq 2$ behave as $\top$. They shall thus be similarly postponed after all other asynchronous rules have been applied. Fortunately, we do not have to worry about disequalities that might only become trivial after the application of substitutions caused by rule permutations, because subderivations will be focused before the whole derivation.

Fixed points also require some preparation of the derivations. In the focused system, a greatest fixed point can be used for co-induction or frozen for later use “as an

atom" in the axiom rule. To reflect this step, we shall enrich our derivations with an extra-logical annotation for frozen fixed points. The heart of our problem is that an instance of ν must be used in the same way in all branches for the focalization process to work. If a greatest fixed point is copied by an additive conjunction, then it should either be used for co-induction in both branches, or frozen and used for axiom in both branches. Otherwise we will fail to permute the treatment of the ν under that of the & while controlling the size of the transformed derivation. We shall instead apply a first *balancing* transformation before the essential part of focalization.

In order to detail the balancing of derivations, we must clarify our notions of formulas and occurrences. Indeed, we need to be able to *identify* several occurrences of a formula, but not necessarily identify all *isomorphic* formulas. To clarify this, we shall use the notion of location — reminiscent of the *loci* from [Gir01]. Intentionally, we remain informal to leave heavy details out of the main discussion, and just try to give the right intuition.

Definition 4.1 (Formula, location, occurrence)**.** A formula shall be seen not only as the structure that it denotes, namely an abstract syntax tree, but also as an address where this structure is located. Similarly, subformulas have their own locations, yielding a *tree of locations* and sublocations. This allows to make a distinction between *identical* formulas, which have the same location, and *isomorphic* formulas which only denote the same structure. In this work, the locations almost always matter. When we talk of the *occurrences* of a formula in a sequent or derivation, we refer to identical formulas occurring in different places in that sequent or derivation.

The locations are independent of the term structure of formulas: all instances of a formula are considered to have the same location. This amounts to say that locations are attached to formulas abstracted over all terms.

Inference rules should be read from the locative viewpoint, the multiset of formulas being in fact a *multiset of locations*. Let us give a couple key examples. In the $\forall$ and $\exists$ rules, the top and bottom sequents only differ in one location: the active location is replaced by its only sublocation. The premise sequents of the $\neq$ rule are locatively identical to the conclusion sequent, except for the location of the active $\neq$ formula that has been removed. In the & rule, the formulas of the context are copied in the two premises, each of them occurring (identically) at least three times at toplevel in the rule. The axiom rule should be read structurally, *i.e.,* it takes place between two locations which denote dual structures.

In the ν rule, the formulas from the co-invariance proofs have *new* locations, as well as the co-invariant in the main premise. This means that these locations can be changed at will, much like a renaming of eigenvariables. In other words, a greatest fixed point has infinitely many sublocations, regarding the coinvariants as its subformulas. In the μ rule, the formula $B(\mu B)\vec{t}$ is the only sublocation of the least fixed point[1]. Notice that in order for the graph of locations to remain a tree, distinct occurrences of μB in $B(\mu B)$ (resp. νB in $B(\nu B)$) must be distinguished.

[1]The details about μ can be safely ignored in a first reading, as they only matter in establishing the completeness of the ν-focused system for μMALL.

Example 4.2. Locations are primarily used to track a formula through a derivation. They also have the important effect of distinguishing fixed points from their recursive occurrences. If *nat* $(s\ x)$ is located at a given address then the recursive occurrence of *nat* in its unfolding, which will eventually become *nat* x, is located at a strict sublocation of that address.

We assume that two formulas of a same sequent have disjoint locations: neither is a sublocation of the other. It is easy to check that this property is preserved by the rules of μMALL.

Definition 4.3 (Freezing-annotated derivation)**.** The freezing-annotated variant of μMALL is obtained by enriching the sequent structure with an annotation for frozen fixed points, restricting the initial rule to be applied only with a frozen greatest fixed point, and adding an explicit annotation rule:

$$\frac{\vdash \Gamma, (\nu B\vec{t})^*}{\vdash \Gamma, \nu B\vec{t}} \qquad \frac{}{\vdash (\nu B\vec{t})^*, \mu\overline{B}\vec{t}}$$

Any μMALL derivation can be transformed into a freezing-annotated one by applying the obvious transformation:

$$\frac{}{\vdash \nu B\vec{t}, \mu\overline{B}\vec{t}} \quad\longrightarrow\quad \frac{\dfrac{}{\vdash (\nu B\vec{t})^*, \mu\overline{B}\vec{t}}}{\vdash \nu B\vec{t}, \mu\overline{B}\vec{t}}$$

From now on we shall work on freezing-annotated derivations, simply calling them derivations.

Definition 4.4 (Balanced derivation)**.** A greatest fixed point occurrence is *used in a balanced way* if all of its principal occurrences are used consistently: either they are all frozen or they are all used for co-induction, with the same co-invariant. We say that a derivation is *balanced* if all greatest fixed points occurring in it are used in a balanced way, and only frozen asynchronous formula are erased in it (by an application of $\top$ or trivial $\neq$).

Lemma 4.5. *If S_0 and S_1 are both coinvariants for B then so is $S_0 \oplus S_1$.*

Proof. Let Π_i be the derivation of coinvariance for S_i. The proof of coinvariance of $S \oplus S'$ is as follows:

$$\frac{\dfrac{\phi_0(\Pi_0)}{S_0\vec{x} \vdash B(S_0 \oplus S_1)\vec{x}} \quad \dfrac{\phi_1(\Pi_1)}{S_1\vec{x} \vdash B(S_0 \oplus S_1)\vec{x}}}{S_0\vec{x} \oplus S_1\vec{x} \vdash B(S_0 \oplus S_1)\vec{x}}$$

The transformed derivations $\phi_i(\Pi_i)$ are obtained by functoriality:

$$\phi_i(\Pi_i) = \frac{\dfrac{\Pi_i}{S_i\vec{x} \vdash BS_i\vec{x}} \quad \dfrac{\dfrac{}{S_i\vec{z} \vdash S_i\vec{z}}}{\dfrac{S_i\vec{z} \vdash S_0\vec{z} \oplus S_1\vec{z}}{BS_i\vec{y} \vdash B(S_0 \oplus S_1)\vec{y}}\ \textit{functo}}}{S_i\vec{x} \vdash B(S_0 \oplus S_1)\vec{x}}\ \textit{cut}$$

Notice that after the elimination of cuts, the proof of coinvariance that we built can be larger than the original ones. □

Lemma 4.6. *Any derivation can be turned into a balanced one.*

Proof. We first ensure that all coinvariants used for the same (locatively identical) greatest fixed points are the same. This is achieved by taking the union of all coinvariants, thanks to Lemma 4.5. We also add to this union the unfolding coinvariant $B(\nu B)$. Let S be the resulting coinvariant, of the form $S_0 \oplus \ldots \oplus S_n \oplus B(\nu B)$, and Θ be the proof of its coinvariance. We adapt our derivation by changing every instance of the ν rule as follows:

$$\dfrac{\vdash \Gamma, S_i\vec{t} \quad \dfrac{\Theta_i}{S_i\vec{x} \vdash BS_i\vec{x}}}{\vdash \Gamma, \nu B\vec{t}} \quad\longrightarrow\quad \dfrac{\dfrac{\vdash \Gamma, S_i\vec{t}}{\vdash \Gamma, S\vec{t}}\oplus \quad \dfrac{\Theta}{S\vec{x} \vdash BS\vec{x}}}{\vdash \Gamma, \nu B\vec{t}}$$

We can assume that the only asynchronous formulas that are erased are frozen fixed points, otherwise this is obtained by applying a finite number of asynchronous rules, including freezing, to postpone the applications of $\top$ and trivial $\neq$. It remains to balance the usage of fixed points, while preserving the property that only frozen asynchronous formulas are erased.

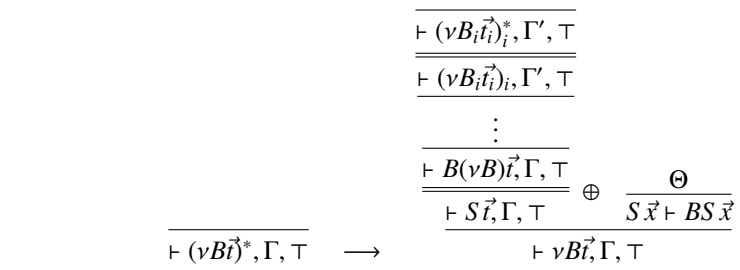
The transformation proceeds by induction on the derivation. Once the subderivations are balanced, they should be glued back to form a balanced derivation. This is non-trivial as soon as one formula occurs in several subderivations, *i.e.,* in the rules & and $\neq$. We shall only consider the case of &, the other one being similar:

$$\dfrac{\dfrac{\Pi}{\vdash \Gamma, \nu B\vec{t}, P} \quad \dfrac{\Xi}{\vdash \Gamma, \nu B\vec{t}, Q}}{\vdash \Gamma, \nu B\vec{t}, P \mathbin{\&} Q}$$

Let Π' and Ξ' denote the balanced versions of Π and Ξ. Although they are balanced, a given $\nu B\vec{t}$ is not necessarily used consistently between them. Let us say that it is frozen in Ξ' but co-inducted on in Π'. We apply the following two transformations on Ξ', replacing all axioms and erasures on $(\nu B\vec{t})^*$: an expansion of the axioms,

$$\dfrac{}{\vdash (\nu B\vec{t})^*, \mu\overline{B}\vec{t}} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B(\nu B)\vec{t}, \overline{B}(\mu\overline{B})\vec{t}}\, init*}{\vdash B(\nu B)\vec{t}, \mu\overline{B}\vec{t}}\,\mu}{\vdash S\vec{t}, \mu\overline{B}\vec{t}}\oplus \quad \dfrac{\Theta}{S\vec{x} \vdash BS\vec{x}}}{\vdash \nu B\vec{t}, \mu\overline{B}\vec{t}}\,\nu$$

and a similar expansion of the erasures[2],



In the expansion of the axiom, the rule *init*∗ denotes the natural adaptation of the derived rule *init* (see Proposition 3.5) to the framework of annotated derivations: if a fixed point formula is encountered, it is frozen and the axiom is applied. In the expansion of the erasure, the vertical dots correspond to treating the outer asynchronous layer of $B(\nu B)\vec{t}$, *i.e.,* applying $\bot, \parr, \&, \forall$ and $\neq$, until only synchronous formulas and greatest fixed points remain. Then, fixed points are frozen and the $\top$ rule can be applied. The same transformation is also used to adapt the erasure of fixed points by an trivial $\neq$.

These two transformations restore a balanced usage for νB, but may introduce freezing on greatest fixed point subformulas of νB: this may occur in *init*∗, but also with the νB_i in our expansion of the erasure. If necessary, *i.e.,* if these subformulas are not frozen in Π', their freezing should be corrected by applying the same transformations on them. At this point the distinction between location and structure (*cf.* Definition 4.1) is crucial: speaking about structure, the subformula relation is cyclic on fixed points (*nat* is a subformula of *nat*); but from the locative point of view it coincides with the tree of sublocations. Moreover, that tree is only finitely explored in Π', so there is a well-founded order on formulas, based on the distance to a leaf in that finite part of the tree. A multiset ordering argument on top of that measure shows that our iterative process terminates. □

Balancing is the most important and novel part of our focalization process. This pre-processing of derivations is only a technical device ensuring termination in the proof of completeness, independently of which rules are permuted down; we stress that it *is* possible in the focused system to apply $\top$ before other asynchronous rules. It is also possible to produce a derivation that is not balanced, *e.g.,* by applying an additive conjunction before treating a greatest fixed point.

4.1.2 Preliminaries

Definition 4.7. We consider the measures $h_\mu(\Pi)$ and $|\Pi|$. $|\Pi|$ is the number of connectives in the conclusion, counting 1 for μ and ν expressions, but 0 for frozen ones. The *μ-height* of a derivation Π with subderivations $(\Pi_i)_i$ is an ordinal, inductively defined by:

$$h_\mu(\Pi) = \begin{cases} 1 + sup\{h_\mu(\Pi_i)\} \text{ if the first rule of } \Pi \text{ is } \mu \text{ or } \nu; \\ sup\{h_\mu(\Pi_i)\} \text{ otherwise.} \end{cases}$$

[2] We could have used a more radical solution by using the coinvariant $\mathbf{0}$. We prefer to invoke the unfolding to show that it works equally well when treating balancing the usages of μ instead of ν.

Proposition 4.8. *The lexicographic order on* $(h_\mu(\Pi), |\Pi|)$ *is compatible with the subderivation order.*

Proof. Any application removes one connective and thus decreases $|\Pi|$ (without changing $h_\mu(\Pi)$), except μ and ν which decrease $h_\mu(\Pi)$. □

The couple $(h_\mu(\Pi), |\Pi|)$ will be simply called the measure or size of a derivation in the followings.

Lemma 4.9. *Let* σ *be a substitution of first-order variables. If* $\vdash \Gamma$ *is provable, then so is* $\vdash \Gamma\sigma$*. Moreover, the instantiated derivation has a least or equal size.*

Proof. This property is a standard and straightforward one, as the fixed points do not change anything here. □

We now present the focalization graph of [MS07].

Definition 4.10. The *synchronous trunk* of a derivation is its largest open subderivation which contains only applications of synchronous rules.

Definition 4.11. We define the relation $\prec$ on the formulas of the base sequent of a derivation Π: $P \prec Q$ if and only if there exists P', asynchronous subformula[3] of P in Π, and Q', synchronous subformula of Q, such that P' and Q' occur in the same sequent of the synchronous trunk of Π.

The intended meaning of $P \prec Q$ is that we must focus on P before Q. Therefore, the natural question is the existence of minimal elements for that relation, equivalent to its acyclicity.

Proposition 4.12. *If* Π *starts with a synchronous rule, and* P *is minimal for* $\prec$ *in* Π*, then so are its subformulas in their respective subderivations.*

Proof. It is enough to notice how the $\prec$ relation evolves in a synchronous trunk. The relations below and on top of a $\oplus$ rule are isomorphic. The same thing holds for $\exists$ and μ. The application of $=$ and $\mathbf{1}$ ends the derivation and hence the synchronous trunk. There only remains the interesting case: the tensor. In that case the relation below the tensor is (isomorphic to) the union of the two relations on top of it, in which only two points get merged, namely the two subformulas of the split tensor. □

Lemma 4.13. *The relation* $\prec$ *is acyclic.*

Proof. We re-use previous observations on the evolution of $\prec$ and proceed by induction on the derivation. There is not anything to do for the cases of $=$ and $\mathbf{1}$. If the derivation starts with a $\oplus$, $\exists$ or μ, the acyclicity on $\prec$ on the conclusion comes from the acyclicity for the subderivation. For $\otimes$, assuming the acyclicity of $\prec$ on the premises, we cannot have a cycle in the conclusion: this cycle cannot lie within the ancestors of a single branch, so it has to involve the split tensor, but then it would have to be involved twice because it is the only node connecting the two ancestors components, and we contradict again the acyclicity of $\prec$ on the premises. □

[3] This does mean subformula in the locative sense, in particular with (co)invariants being subformulas of the associated fixed points.

4.1.3 Permutation lemmas and completeness

Lemma 4.14. *Let P be an asynchronous formula. Any balanced derivation of $\vdash \Gamma, P$ can be turned into one where P is active in the conclusion, which is still balanced and which has a smaller or equal size than the original.*

Proof. We proceed by induction on the derivation. If P is not active in the first rule, then by induction hypothesis make it active in the immediate subderivations where it occurs. Then permute the first two rules. A simple inspection shows that in each case, the resulting derivation has equal (in most of the cases) or smaller size (*e.g.,* when permuting down a $\top$ rule), and that balancing is not affected.

The MALL permutations are usual and interact well with our measure. Most of the permutations involving the new rules are not surprising, such as $\otimes/\nu$:

$$\dfrac{\dfrac{\vdash \Gamma, P, S\vec{t} \quad \vdash BS\vec{x}, S\vec{x}^{\perp}}{\vdash \Gamma, P, \nu B\vec{t}} \qquad \vdash \Gamma', P'}{\vdash \Gamma, \Gamma', P \otimes P', \nu B\vec{t}} \quad \longrightarrow \quad \dfrac{\dfrac{\vdash \Gamma, P, S\vec{t} \quad \vdash \Gamma', P'}{\vdash \Gamma, \Gamma', P \otimes P', S\vec{t}} \qquad \vdash BS\vec{x}, S\ \vec{x}^{\perp}}{\vdash \Gamma, \Gamma', P \otimes P', \nu B\vec{t}}$$

The $\&/\nu$ permutation relies on the fact that our derivation is balanced:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi}{\vdash \Gamma, P, S\vec{t}} \quad \dfrac{\Theta}{\vdash BS\vec{x}, S\vec{x}^{\perp}}}{\vdash \Gamma, P, \nu B\vec{t}} \qquad \dfrac{\dfrac{\Pi'}{\vdash \Gamma, P', S\vec{t}} \quad \dfrac{\Theta}{\vdash BS\vec{x}, S\vec{x}^{\perp}}}{\vdash \Gamma, P', \nu B\vec{t}}}{\vdash \Gamma, P \,\&\, P', \nu B\vec{t}}}{\Big\downarrow}$$

$$\dfrac{\dfrac{\dfrac{\Pi}{\vdash \Gamma, P, S\vec{t}} \quad \dfrac{\Pi'}{\vdash \Gamma, P, S'\vec{t}}}{\vdash \Gamma, P \,\&\, P', S\vec{t}} \qquad \dfrac{\Theta}{\vdash BS\vec{x}, (S\vec{x})^{\perp}}}{\vdash \Gamma, P \,\&\, P', \nu B\vec{t}}$$

Another non-trivial case is $\otimes/\neq$ which makes use of Lemma 4.9:

$$\dfrac{\dfrac{\{\vdash (\Gamma, P)\sigma : \sigma \in csu(u, v)\}}{\vdash \Gamma, P, (u = v)^{\perp}} \qquad \vdash \Gamma', Q}{\vdash \Gamma, \Gamma', P \otimes Q, (u = v)^{\perp}} \quad \rightarrow \quad \left\{ \dfrac{\dfrac{\vdash (\Gamma, P)\sigma \quad \vdash (\Gamma', Q)\sigma}{\vdash (\Gamma, \Gamma', P \otimes Q)\sigma} \quad : \sigma \in csu(u, v)}{\vdash \Gamma, \Gamma', P \otimes Q, (u = v)^{\perp}} \right\}$$

□

Lemma 4.15. *Let Γ be a sequent of synchronous formulas. If it has a derivation, and P is minimal for $\prec$ in the conclusion of that derivation, there there is also a derivation where P is active. Moreover, the new derivation has a smaller or equal size, P is still minimal in its conclusion, and balancing is preserved.*

Proof. If P is already active, there is nothing to do. Otherwise, we will make it active in the subderivations. We can do that by induction: by minimality of P the subderivation's conclusion does not have any asynchronous formula, and P is still minimal in the subderivations by Proposition 4.12. Then we permute the first two layers of rules, which are synchronous. The permutation of synchronous rules are already known for

MALL, and the new cases involving = or μ are straightforward. Moreover the permutation of synchronous rules within the synchronous trunk does not affect minimality, which only depends on the frontier of the trunk. The preservation of size and balancing is easy to check. □

Theorem 4.16. *The μ-focused system is sound and complete with respect to μMALL.*

Proof. Soundness is trivial. We prove completeness by induction on the size of the balanced derivation:

- If there is any, pick an asynchronous formula *arbitrarily*, and transform the derivation by making that formula active thanks to Lemma 4.14. By induction, focus the subderivations, and add the first rule in the focused system.
- When there is no asynchronous formula left, we've shown in Lemma 4.13 that there is a minimal synchronous formula. We can then apply Lemma 4.15 to make that formula active. While there are no asynchronous formulas in the conclusion, we repeat that step, choosing the subformula of the previous minimal formula among the new minimal ones (Proposition 4.12). This strategy translates well to hereditary rule applications in the focused system.

□

4.2 Application to μLJL

The examples of Section 3.4 showed that despite its simplicity and linearity, μMALL can be related to a more conventional logic. In particular we are interested in drawing some connections with μLJ. In the following, we show a simple first step to this program, in which we actually capture a non-trivial fragment of μLJ even though μMALL does not have exponentials at all.

We have observed (Proposition 6.11) that structural rules are admissible for negative formulas of μMALL, which coincide with fully asynchronous formulas in the μ-focusing system. This property allows us to obtain a faithful encoding of a fragment of μLJ in μMALL despite the absence of exponentials. The encoding must be organized so that formulas appearing on the left-hand side of intuitionistic sequents can be encoded positively in μMALL. The only connectives allowed to appear negatively shall thus be $\wedge$, $\vee$, =, μ and $\exists$. Moreover, the encoding must commute with negation, in order to translate the (co)induction rules correctly. This leaves no choice in the following design.

Definition 4.17 (μLJL)**.** The logic μLJL is the restriction of μLJ to sequents where all hypothesis are in the fragment $\mathcal{H}$, and the goal is in the fragment $\mathcal{G}$. These fragments are given by:

$$\begin{array}{rcl} \mathcal{G} & ::= & \mathcal{G} \wedge \mathcal{G} \mid \mathcal{G} \vee \mathcal{G} \mid s = t \mid \exists x.\mathcal{G}x \mid \mu(\lambda p \lambda \vec{x}.\mathcal{G})\vec{t} \mid p\vec{t} \\ & \mid & \forall x.\mathcal{G}x \mid \mathcal{H} \supset \mathcal{G} \mid \nu(\lambda p \lambda \vec{x}.\mathcal{G})\vec{t} \\ \mathcal{H} & ::= & \mathcal{H} \wedge \mathcal{H} \mid \mathcal{H} \vee \mathcal{H} \mid s = t \mid \exists x.\mathcal{H}x \mid \mu(\lambda p \lambda \vec{x}.\mathcal{H})\vec{t} \mid p\vec{t} \end{array}$$

Formulas in $\mathcal{H}$ and $\mathcal{G}$ are translated in μMALL as follows:

$$\begin{array}{rcl@{\qquad}rcl}
[P \wedge Q] & \stackrel{def}{=} & [P] \otimes [Q] & [\forall x.Px] & \stackrel{def}{=} & \forall x.[Px] \\
[P \vee Q] & \stackrel{def}{=} & [P] \oplus [Q] & [\nu B\vec{t}] & \stackrel{def}{=} & \nu[B]\vec{t} \\
[s = t] & \stackrel{def}{=} & s = t & [P \supset Q] & \stackrel{def}{=} & [P] \multimap [Q] \\
[\exists x.Px] & \stackrel{def}{=} & \exists x.[Px] & [\lambda p\lambda\vec{x}.Bpx] & \stackrel{def}{=} & \lambda p\lambda\vec{x}.[Bpx] \\
[\mu B\vec{t}] & \stackrel{def}{=} & \mu[B]\vec{t} & [p\vec{t}] & \stackrel{def}{=} & p\vec{t}
\end{array}$$

Proposition 4.18. *For any $P \in \mathcal{G}$, P is provable in μLJ if and only if $[P]$ is provable in μMALL, under the restrictions that (co)invariants $\lambda\vec{x}.S\,\vec{x}$ in μMALL (resp. μLJ) are such that $S\,\vec{x}$ is in $[\mathcal{H}]$ (resp. $\mathcal{H}$).*

Proof. The proof transformations are simple and compositional. The induction rule corresponds to the ν rule for $(\mu B\vec{t})^{\perp}$, the proviso on invariants allowing the translations:

$$\frac{\Gamma, S\,\vec{t} \vdash G \quad BS\,\vec{x} \vdash S\,\vec{x}}{\Gamma, \mu B\vec{t} \vdash G} \qquad \longleftrightarrow \qquad \frac{[\Gamma], [S\,\vec{t}] \vdash [G] \quad [B][S\,]\vec{x} \vdash [S\,\vec{x}]}{[\Gamma], \mu[B]\vec{t} \vdash [G]}$$

This relies on a few simple observations: $[B][S\,]\vec{x}$ is the same as $[BS\,\vec{x}]$; and if both B and S are in $\mathcal{H}$ then so is BS, hence we remain in our fragment. The same goes for coinduction, except that in that case B can be in $\mathcal{G}$:

$$\frac{\Gamma \vdash S\,\vec{t} \quad S\,\vec{x} \vdash BS\,\vec{x}}{\Gamma \vdash \nu B\vec{t}} \qquad \longleftrightarrow \qquad \frac{[\Gamma] \vdash [S\,]\vec{t} \quad [S\,\vec{x}] \vdash [BS\,\vec{x}]}{[\Gamma] \vdash \nu[B]\vec{t}}$$

In order to restore the additive behavior of some intuitionistic rules (*e.g.*, $\wedge R$) and translate the structural rules, we can contract and weaken our negative μMALL formulas corresponding to μLJL hypothesis. □

Linear logic provides an appealing proof theoretic setting because of its emphasis on dualities and of its clear separation of concepts (additive/multiplicative, asynchronous/synchronous). Our experience is that μMALL is a good place to study focusing in the presence of least and greatest fixed point connectives. To get similar results for μLJ, one can either work from scratch entirely within the intuitionistic framework or use an encoding into linear logic. Given a mapping from intuitionistic to linear logic, and a complete focused proof system for linear logic, one can often build a complete "focalized" proof-system for intuitionistic logic.

$$\begin{array}{ccc}
\vdash F & \longrightarrow & \vdash [F] \\
\vdots & & \downarrow \\
\vdash\Uparrow F & \longleftarrow & \vdash\Uparrow [F]
\end{array}$$

The usual encoding of intuitionistic logic into linear logic involves exponentials, which can damage focusing structures (by causing both synchronous and asynchronous phases to end). Hence, a careful study of the polarity of linear connectives must be

done (cf. [DJS93, LM07a]) in order to minimize the role played by the exponentials in such encodings. Here, as a result of Proposition 4.18, it is possible to get a complete focused system for μLJL that inherits precisely the strong structure of the linear μ-focused derivations.

This system is presented in Figure 4.2. Its sequents have the form $\Gamma; \Gamma' \vdash P$ where Γ' is a multiset of synchronous formulas (fragment $\mathcal{H}$) and the set Γ contains frozen least fixed points. First, notice that accordingly with the absence of exponential in the encoding into linear logic, there is no structural rule. The asynchronous phase takes place on sequents where Γ is not empty. Indeed, Γ will only contain left-asynchronous formulas. The synchronous phase processes sequents of the form $\Gamma; \vdash P$, where the focus is without any ambiguity on P. Notice that it is impossible to introduce any connective on the right when Γ' is not empty. Strictly speaking, the synchronous phase actually contains some asynchronous rules: implication, universal quantification and coinduction. We ignore this in order to simplify the presentation, which is harmless since there is no choice in refocusing afterwards. Finally, notice that the linear encoding tells us that we could even require Γ to be treated linearly, be empty in the initial rule for equality and have exactly one element in the initial rule for fixed points. Indeed, fixed points in $\mathcal{H}$ can be explicitly weakened if needed. But we relaxed this constraint as it does not seem to make a better system.

Proposition 4.19. *The focused proof system for μLJL is sound and complete with respect to μLJL.*

Proof. As explained above, that focused system follows directly from the constraints imposed by the μ-focused system of μMALL on the translations of μLJL derivations. □

Although μLJL is only a small fragment of μLJ, it catches many interesting and useful problems. For example, any Horn-clause specification can be expressed in $\mathcal{H}$ as a least fixed point and theorems that state properties such as totality or functionality of predicates defined in this manner are in $\mathcal{G}$. Theorems that state more model-checking properties, for example, $\forall x.p(x) \supset q(x)$, where p and q are one-placed least fixed point expressions over $[H]$, are also in $\mathcal{G}$. Finally, the theorems about natural numbers presented in Section 3.4 are within $[\mathcal{G}]$. However, two of the proofs of these theorems (for the totality of $half$ and that the sum of natural numbers is a natural number) do not satisfy the restriction on co-invariants. We shall see, however, that there is a simpler proof for the totality of $half$ that is in μLJL. More generally, we shall define in Chapter 5 a general class of formulas for which we obtain a completeness result within μLJL.

The logic μLJ is closely related to LINC [Tiu04]. The main difference is the absence of the ∇ quantifier in our system, but we show in Chapter 6 that it could be added in an orthogonal fashion. The resulting extension to μMALL (and μLJL) should allow natural ways to reason about specifications involving variable bindings, in the manner illustrated in [BGM$^+$07b, Tiu04, Tiu05]. Interestingly, the fragment $\mathcal{G}$ has already been identified in LINC [TNM05], and the Bedwyr system [BGM$^+$07b] presented in Chapter 7 implements a proof-search strategy for it that is complete under the assumption that all fixed points are noetherian. This strategy coincides with the focused system for μLJL restricted to unfoldings, discarding freezing and general (co)induction. Under

Asynchronous phase

$$\frac{\Gamma;\Gamma', P, Q \vdash R}{\Gamma;\Gamma', P \wedge Q \vdash R} \quad \frac{\Gamma;\Gamma', P \vdash R \quad \Gamma;\Gamma', Q \vdash R}{\Gamma;\Gamma', P \vee Q \vdash R}$$

$$\frac{\Gamma;\Gamma', Pc \vdash Q}{\Gamma;\Gamma', \exists x.Px \vdash Q}$$

$$\frac{\{(\Gamma;\Gamma' \vdash P)\theta : \theta \in csu(s \doteq t)\}}{\Gamma;\Gamma', s = t \vdash P}$$

$$\frac{\Gamma, \mu B\vec{t};\Gamma' \vdash P}{\Gamma;\Gamma', \mu B\vec{t} \vdash P} \quad \frac{\Gamma;\Gamma', S\vec{t} \vdash P \quad BS\vec{x} \vdash S\vec{x}}{\Gamma;\Gamma', \mu B\vec{t} \vdash P}$$

Synchronous phase

$$\frac{\Gamma;\vdash A \quad \Gamma;\vdash B}{\Gamma;\vdash A \wedge B} \quad \frac{\Gamma;\vdash A_i}{\Gamma;\vdash A_0 \vee A_1} \quad \frac{\Gamma;A \vdash B}{\Gamma;\vdash A \supset B}$$

$$\frac{}{\Gamma;\vdash t = t} \quad \frac{\Gamma;\vdash Pt}{\Gamma;\vdash \exists x.Px} \quad \frac{\Gamma;\vdash Pc}{\Gamma;\vdash \forall x.Px}$$

$$\frac{}{\Gamma, \mu B\vec{t};\vdash \mu B\vec{t}} \quad \frac{\Gamma;\vdash B(\mu B)\vec{t}}{\Gamma;\vdash \mu B\vec{t}}$$

$$\frac{S \in \mathcal{H} \quad \Gamma;\vdash S\vec{t} \quad S\vec{x} \vdash BS\vec{x}}{\Gamma;\vdash \nu B\vec{t}}$$

Figure 4.2: Focused proof system for μLJL

such restrictions, there is no need for any contraction and one can always eagerly eliminate left-hand side (asynchronous) connectives before working on the goal (right-hand side); in particular, there is no need for the initial rule $\mu\nu$.

4.3 The ν-focused system

While the classification of μ as synchronous and ν as asynchronous is rather satisfying and coincides with several other observations, that choice does not seem to be forced from the focusing point of view alone. After all, the μ rule also commutes with all other rules. It turns out that one can design a ν-focused system treating μ as asynchronous and ν as synchronous, and still obtain completeness. That system is presented in Figure 4.3.

Asynchronous phase

$$\frac{\vdash \Gamma \Uparrow P, Q, \Delta}{\vdash \Gamma \Uparrow P \parr Q, \Delta} \qquad \frac{\vdash \Gamma \Uparrow P, \Delta \quad \vdash \Gamma \Uparrow Q, \Delta}{\vdash \Gamma \Uparrow P \,\&\, Q, \Delta}$$

$$\frac{\vdash \Gamma \Uparrow \Delta}{\vdash \Gamma \Uparrow \bot, \Delta} \qquad \frac{}{\vdash \Gamma \Uparrow \top, \Delta} \qquad \frac{\{\vdash \Gamma\theta \Uparrow \Delta\theta : \theta \in csu(s \doteq t)\}}{\vdash \Gamma \Uparrow s \neq t, \Delta}$$

$$\frac{\vdash \Gamma \Uparrow Pc, \Delta}{\vdash \Gamma \Uparrow \forall x.Px, \Delta}\ c \text{ new}$$

$$\frac{\vdash \Gamma \Uparrow B(\mu B)\vec{t}, \Delta}{\vdash \Gamma \Uparrow \mu B\vec{t}, \Delta} \qquad \frac{\vdash \Gamma, \mu B\vec{t} \Uparrow \Delta}{\vdash \Gamma \Uparrow \mu B\vec{t}, \Delta}$$

Synchronous phase

$$\frac{\vdash \Gamma \Downarrow P \quad \vdash \Gamma' \Downarrow Q}{\vdash \Gamma, \Gamma' \Downarrow P \otimes Q} \qquad \frac{\vdash \Gamma \Downarrow P_i}{\vdash \Gamma \Downarrow P_0 \oplus P_1}$$

$$\frac{}{\vdash \Downarrow \mathbf{1}} \qquad \frac{}{\vdash \Downarrow t = t}$$

$$\frac{\vdash \Gamma \Downarrow Pt}{\vdash \Gamma \Downarrow \exists x.Px}$$

$$\frac{\vdash \Gamma \Downarrow S\vec{t} \quad \vdash \Uparrow BS\vec{x}, (S\vec{x})^{\perp}}{\vdash \Gamma \Downarrow \nu B\vec{t}} \qquad \frac{}{\vdash \mu\overline{B}\vec{t} \Downarrow \nu B\vec{t}}$$

Switching (where P is synchronous, Q asynchronous)

$$\frac{\vdash \Gamma, P \Uparrow \Delta}{\vdash \Gamma \Uparrow P, \Delta} \qquad \frac{\vdash \Gamma \Downarrow P}{\vdash \Gamma, P \Uparrow} \qquad \frac{\vdash \Gamma \Uparrow Q}{\vdash \Gamma \Downarrow Q}$$

Figure 4.3: The ν-focused proof-system for μMALL

Theorem 4.20. *The ν-focused system is sound and complete with respect to μMALL.*

Proof. The proof follows the same argument as for the μ-focused system. First, the usage of least fixed points can be balanced: we took care in the previous proof to make an argument for ν that would work equally well for μ. It is in fact simpler for μ, as there is no need to merge all possible coinvariants: the balancing process is only ensuring that a least fixed point is unfolded the same number of times in each branch of the proof. Then, balanced proofs can be put in focused form by permuting rules. The permutations are as before, the non-trivial step of permuting a μ and a & being made possible by balancing. □

This flexibility in the design of a focusing system is surprising. It is not of the same nature as the arbitrary bias assignment that can be done in Andreoli's system: atoms are non-canonical, and the bias can be seen as a way to indicate what is the synchrony of the formula that a given atom might be instantiated with. Here, our fixed points are fully defined and canonical. This flexibility highlights the fact that focusing is a somewhat shallow property, only accounting for local rule permutations independently of deeper properties like positivity. Although we do not see any practical use of such flexibility, it is not excluded that one is discovered in the future, much like with the arbitrary bias assignment on atoms in Andreoli's original system.

To go one step further, we could consider an arbitrary "bias" assignment to fixed point formulas, to specify if they should behave synchronously or asynchronously. We would consider two logically identical variants of each fixed point: μ^+ and ν^+ being treated synchronously, μ^- and ν^- asynchronously. It turns out to be difficult to ensure the termination of the balancing process in that case: in order to balance a ν^-, one may have to unfold a μ^- and possibly break its balanced use, and vice versa. A simple and natural way to ensure the termination of balancing is to restrict the axiom rule to take place between fixed points of opposite bias:

$$\frac{}{\vdash (\mu B\vec{t})^+, (\nu\overline{B}\vec{t})^-} \qquad \frac{}{\vdash (\nu B\vec{t})^+, (\mu\overline{B}\vec{t})^-}$$

While we would then be able to design a complete focused system with respect to μMALL with bias, it is important to notice that biased μMALL is not complete with respect to μMALL.

4.4 Exponentials and μLJ

We have defined two focusing systems for μMALL, based on a common methodology. In this last section, we show that our proof also naturally extends to exponentials. Considering exponentials is interesting regarding the flexibility in the synchrony of fixed points: the question is whether the focusing treatment of exponentials requires synchrony and positivity to coincide. We shall see that it is not the case. Finally, we design a focusing system for μLJ and see why the usual techniques used to carry focusing from linear to intuitionistic logic do not apply here.

4.4.1 μLL

Let us consider the μ-focused system for μLL in Figure 4.4. That system treats exponentials as in Andreoli's focusing system for LL (Figure 1.3). It also uses the same dyadic sequents, with a new non-linear zone Θ. The $?P$ formulas enter that zone in the asynchronous phase. Formulas in Θ can be focused on, which triggers their contraction-dereliction. Finally, $!P$ is introduced in the synchronous phase, which has to release the focus. The proof technique of [MS07, Sau08] has been shown to extend naturally to that treatment of exponentials. It remains true in our case.

Asynchronous phase

$$\frac{\vdash \Theta; \Gamma \Uparrow P, Q, \Delta}{\vdash \Theta; \Gamma \Uparrow P \parr Q, \Delta} \qquad \frac{\vdash \Theta; \Gamma \Uparrow P, \Delta \quad \vdash \Theta; \Gamma \Uparrow Q, \Delta}{\vdash \Theta; \Gamma \Uparrow P \,\&\, Q, \Delta}$$

$$\frac{\vdash \Theta; \Gamma \Uparrow \Delta}{\vdash \Theta; \Gamma \Uparrow \bot, \Delta} \qquad \frac{}{\vdash \Theta; \Gamma \Uparrow \top, \Delta} \qquad \frac{\vdash \Theta, P; \Gamma \Uparrow \Delta}{\vdash \Theta; \Gamma \Uparrow ?P, \Delta}$$

$$\frac{\{(\vdash \Theta; \Gamma \Uparrow \Delta)\theta : \theta \in csu(s \doteq t)\}}{\vdash \Theta; \Gamma \Uparrow s \neq t, \Delta} \qquad \frac{\vdash \Theta; \Gamma \Uparrow Pc, \Delta}{\vdash \Theta; \Gamma \Uparrow \forall x.Px, \Delta}\ c \text{ new}$$

$$\frac{\vdash \Theta; \Gamma \Uparrow S\vec{t}, \Delta \quad \vdash ; \Uparrow BS\vec{x}, S\vec{x}^{\perp}}{\vdash \Theta; \Gamma \Uparrow \nu B\vec{t}, \Delta}\ \vec{x} \text{ new} \qquad \frac{\vdash \Theta; \Gamma, \nu B\vec{t} \Uparrow \Delta}{\vdash \Theta; \Gamma \Uparrow \nu B\vec{t}, \Delta}$$

Synchronous phase

$$\frac{\vdash \Theta; \Gamma \Downarrow P \quad \vdash \Theta; \Gamma' \Downarrow Q}{\vdash \Theta; \Gamma, \Gamma' \Downarrow P \otimes Q} \qquad \frac{\vdash \Theta; \Gamma \Downarrow P_i}{\vdash \Theta; \Gamma \Downarrow P_0 \oplus P_1}$$

$$\frac{}{\vdash \Theta; \Downarrow \mathbf{1}} \qquad \frac{\vdash \Theta; \Gamma \Uparrow P}{\vdash \Theta; \Gamma \Downarrow !P}$$

$$\frac{}{\vdash \Theta; \Downarrow t = t} \qquad \frac{\vdash \Theta; \Gamma \Downarrow Pt}{\vdash \Theta; \Gamma \Downarrow \exists x.Px}$$

$$\frac{\vdash \Theta; \Gamma \Downarrow B(\mu B)\vec{x}}{\vdash \Theta; \Gamma \Downarrow \mu B\vec{x}} \qquad \frac{}{\vdash \Theta; \nu\overline{B}\vec{t} \Downarrow \mu B\vec{t}} \qquad \frac{}{\vdash \Theta, \nu\overline{B}\vec{t}; \Downarrow \mu B\vec{t}}$$

Switching (where P is synchronous, Q asynchronous)

$$\frac{\vdash \Theta; \Gamma, P \Uparrow \Delta}{\vdash \Gamma; \Gamma \Uparrow P, \Delta} \qquad \frac{\vdash \Theta; \Gamma \Downarrow P}{\vdash \Theta; \Gamma, P \Uparrow} \qquad \frac{\vdash \Theta, P; \Gamma \Downarrow P}{\vdash \Theta, P; \Gamma \Uparrow} \qquad \frac{\vdash \Theta; \Gamma \Uparrow Q}{\vdash \Theta; \Gamma \Downarrow Q}$$

Figure 4.4: The μ-focused proof-system for μLL

Theorem 4.21. *The μ-focused system for μLL is sound and complete with respect to μLL.*

Proof. We explain how to adapt our proof. First, it is important to precise the locative reading of contraction-derelictions: each contracted formula is given a different new

sublocation. For example, say that if l is the location of $?P$, its n^{th} copy has location $\langle l, n\rangle$. It is not true anymore that dyadic sequents only contain disjoint locations, but that property still holds for their linear zone; all what is ensured for the whole sequent is that it does not contain twice the same location (at toplevel).

Balancing adapts well to this treatment of locations, the non-trivial steps being the same as before, *i.e.,* & and $\neq$. Since contractions of greatest fixed points are differentiated, they are not tied in the balancing process but can be used in different ways.

The notion of synchronous trunk is adapted to contain the new synchronous rule, namely contraction-dereliction, but also the promotion rule. However, the trunk stops after the promotion rule, whose premise is a leaf of the open derivation. This irregularity is necessary to obtain the necessary permutations within the trunk since no rule permutes with the promotion rule, and suffices because the focus is released on the promotion rule in the focusing system.

It is easy to check that the precedence relation ($\prec$) defined on the sequents of synchronous trunks is still acyclic. Finally, we adapt the definition of the size of derivations by adding a component counting the number of contractions, and establish size-preserving permutation lemmas. The new permutations are straightforward: we have to check that the ? rule can be permuted down, and that the synchronous rules (including contraction-dereliction) commute. The obtained lemmas can finally be combined as before to obtain a translation of derivations to focused derivations. □

It is straightforward to adapt this proof to obtain the completeness of a ν-focused variant for μLL. Indeed, this focusing treatment of exponential still solely relies on rule permutations, which does not discriminate between the two fixed point connectives.

However, there are systems where focusing and exponentials are more entangled. Inspired by LU [Gir91b], these systems are based on derivations where introduction rules can take place inside the non-linear zone. If P and Q are negative, then so is $P \parr Q$ and thus the following rule is admissible:

$$\frac{\vdash \Theta, P, Q; \Gamma}{\vdash \Theta, P \parr Q; \Gamma}$$

It is also invertible. Generalizing that idea, all negative formulas can have their connectives introduced in the non-linear zone, and one can forbid any contraction-dereliction on negative formulas. These observations can lead to focused system where the exponentials interfere less with the focus. They are interesting in that it removes useless formulas from the non-linear zone, and also makes the economy of round-trips through the linear zone. A good example[4] of this kind of system is LJF [LM07a]. In such a focusing system, the polarity of fixed points, and not only their synchrony, would finally have to be taken into account.

It is easy to see that the following rules are admissible:

$$\frac{\vdash \Theta, (\nu B\vec{t})^*; \Gamma}{\vdash \Theta, \nu B\vec{t}; \Gamma} \qquad \frac{\vdash \Theta, S\,\vec{t}; \Gamma \quad \vdash; (S\,\vec{x})^\perp, BS\,\vec{x}}{\vdash \Theta, \nu B\vec{t}; \Gamma}$$

[4] In LJF, the refinement of the treatment of exponentials is based on that idea but uses a translation of formulas that removes redundant exponentials instead of expressing this as a rule of the underlying focused linear logic. Such a static elimination of redundant exponentials is not possible with fixed points.

To show that these are asynchronous rules, *i.e.,* that they can be permuted under other rules, it suffices to observe that the balancing argument can be strengthened a lot: it is possible to require that all contractions of a greatest fixed point are used consistently. Shockingly, we did not have to rely on the polarity of νB — which depends on that of B. Hence, we conjecture a system where these two rules would be the only ones available for a greatest fixed point in the non-linear zone, notably excluding its dereliction, independently of its polarity. The only difference in such a system between a synchronous and an asynchronous body occurs after the unfolding: only in the latter case is it possible to keep working in the non-linear zone. We do not detail further the design and justification of such a system, as it would not bring much and is currently unsatisfying and insufficiently understood.

4.4.2 Focusing μLJ

Before studying the full logic μLJ, let us restrict our attention to the fragment νLJL, which is obtained from μLJL by exchanging the roles of μ and ν. The design of μLJL relied on the positivity of μ, but this only makes sense from the linear point of view. The question is: in the intuitionistic framework, is there a dissymmetry between μ and ν? The answer seems to be negative as far we can tell from the observation of νLJL; the notable point is that μMALL cannot explain it.

Definition 4.22 (νLJL)**.** The logic νLJL is the restriction of μLJ to sequents where all hypothesis are in the fragment $\mathcal{H}$, and the goal is in the fragment $\mathcal{G}$. These fragments are given by:

$$\begin{array}{rcl} \mathcal{G} & ::= & \mathcal{G} \wedge \mathcal{G} \mid \mathcal{G} \vee \mathcal{G} \mid s = t \mid \mu(\lambda p \lambda \vec{x}.\mathcal{G})\vec{t} \mid \exists x.\mathcal{G}x \\ & \mid & \forall x.\mathcal{G}x \mid \mathcal{H} \supset \mathcal{G} \mid \nu(\lambda p \lambda \vec{x}.\mathcal{G})\vec{t} \\ \mathcal{H} & ::= & \mathcal{H} \wedge \mathcal{H} \mid \mathcal{H} \vee \mathcal{H} \mid s = t \mid \nu(\lambda p \lambda \vec{x}.\mathcal{H})\vec{t} \mid \exists x.\mathcal{H}x \end{array}$$

We consider the focusing system for νLJL given in Figure 4.4.2, naively obtained as the symmetric of the focusing system for μLJL. It turns out to be complete, which can be proved using the same tools as before, obtaining a strong balancing by identifying the locations of all contractions of an hypothesis. Strictly speaking, that focusing system is not really the exact symmetric of the other: for μLJL, the non-linear treatment of the frozen zone was allowed for convenience only; here, it is an essential ingredient of completeness. We provide a direct proof of completeness of that focused system, before discussing why it cannot be explained from a linear reading.

Proposition 4.23. *The focused system in νLJL is sound and complete with respect to νLJL.*

Proof. The proof follows the same outline as before. We consider unfocused proofs with an explicit freezing annotation, and reorganize them to obtain a focused structure. The proof relies on the observation that we can obtain a strong balancing, enforcing that contractions of a greatest fixed point are used consistently. It is then possible to permute down all left rules. The case of greatest fixed point rules (coinduction and freezing) is where balancing is exploiting. In the end, right rules and contractions are

Asynchronous phase

$$\frac{\Gamma;\Gamma', P, Q \vdash R}{\Gamma;\Gamma', P \wedge Q \vdash R} \quad \frac{\Gamma;\Gamma', P \vdash R \quad \Gamma;\Gamma', Q \vdash R}{\Gamma;\Gamma', P \vee Q \vdash R}$$

$$\frac{\Gamma;\Gamma', Pc \vdash Q}{\Gamma;\Gamma', \exists x.Px \vdash Q}$$

$$\frac{\{(\Gamma;\Gamma' \vdash P)\theta : \theta \in csu(s \doteq t)\}}{\Gamma;\Gamma', s = t \vdash P}$$

$$\frac{\Gamma, \nu B\vec{t};\Gamma' \vdash P}{\Gamma;\Gamma', \nu B\vec{t} \vdash P} \quad \frac{\Gamma;\Gamma', B(\nu B)\vec{t} \vdash P}{\Gamma;\Gamma', \nu B\vec{t} \vdash P}$$

Synchronous phase

$$\frac{\Gamma;\vdash A \quad \Gamma;\vdash B}{\Gamma;\vdash A \wedge B} \quad \frac{\Gamma;\vdash A_i}{\Gamma;\vdash A_0 \vee A_1} \quad \frac{\Gamma;A \vdash B}{\Gamma;\vdash A \supset B}$$

$$\frac{}{\Gamma;\vdash t = t} \quad \frac{\Gamma;\vdash Pt}{\Gamma;\vdash \exists x.Px} \quad \frac{\Gamma;\vdash Pc}{\Gamma;\vdash \forall x.Px}$$

$$\frac{}{\Gamma, \mu B\vec{t};\vdash \mu B\vec{t}} \quad \frac{\Gamma;\vdash B(\mu B)\vec{t}}{\Gamma;\vdash \mu B\vec{t}}$$

$$\frac{S \in \mathcal{H} \quad \Gamma;\vdash S\vec{t} \quad S\vec{x} \vdash BS\vec{x}}{\Gamma;\vdash \nu B\vec{t}}$$

Figure 4.5: Focused proof system for νLJL

only applied when the left hand-side is restricted to frozen fixed points: we have a focused derivation. □

Unlike μLJL, the formulas of νLJL cannot be encoded in μMALL, because the hypothesis fragment $\mathcal{H}$ allows greatest fixed points, which are not positive. Let us consider encoding it in μLL. It is probably not enough to only use an exponential when encoding the implication, *i.e.,* $[H \supset G] \equiv (![H]) \multimap [G]$, because one would still lack the ability to weaken subformulas of derelicted copies of hypothesis. Even then, we see that the focusing behavior obtained from the linear reading is poor: any introduction on the left requires a copy and renders its active formula linear. In particular, any work on the left hand-side has to be duplicated over branches of a conjunction. A proper encoding would have to use exponentials inside the encoding of the fragment $\mathcal{H}$, notably inside fixed point bodies. Then, we could use the observation that ν can be introduced directly within the non-linear zone, obtaining a very simple focusing behavior. Unfortunately, there is no symmetric explanation for ! and μ, so the focusing on conclusions would still be unsatisfactory.

If linear logic cannot (yet) give a clear account of intuitionistic focusing treatments of fixed points for the νLJL fragment, there is no hope to do it for μLJ. We propose a μ-focused system for μLJ, in Figure 4.6, and claim its completeness, as well as that of the ν-focused variant. This system is deliberately simple, and does not give an account of alternative ways to focus μLJ; our goal is only to give the definition of the focusing system used as a basis for the automated proof-search tactic developed in the Taci system, which we describe in Chapter 7. In particular we do not consider synchronous and asynchronous variants of the conjunction like [LM07a], and restrict to the synchronous one. In an attempt to keep things readable, our focused system only uses extra annotations in the synchronous phase, the asynchronous phase simply being a saturation of all possible asynchronous rules. The synchronous sequents $\Gamma \Downarrow P \vdash Q$ and $\Gamma \vdash \Downarrow P$ respectively denote left and right focus on P.

4.5 Conclusion

We have designed a μ-focused system for μMALL and established its completeness. This system is satisfying in the way that it constrains proofs, and the synchronous treatment of μ coincides with several other theoretical observations: the positive nature of μ (Proposition 6.11), but also game theoretic semantics [MS06]. The μ-focusing, together with the positivity of μ, also allowed for an elegant connection with μLJL, a simple fragment of μLJ that had already been identified in the practical context of logic programming with fixed points.

Then, we have described several ramifications of our study of focusing. We have seen that despite the positive polarity of μ, there is a complete focused system that treats that connective as asynchronous. We have shown how our proof extends to exponentials in the traditional way, and that the flexibility in the synchrony of fixed points persists in that setting. We gave a focused proof system for μLJ, and explained how several phenomenons remained to be understood to clearly connect the linear and intuitionistic focused systems.

Asynchronous phase

$$\frac{\Gamma \vdash Q}{\Gamma, \top \vdash Q} \quad \frac{\Gamma, P, P' \vdash Q}{\Gamma, P \wedge P' \vdash Q} \quad \frac{\Gamma, P \vdash Q \quad \Gamma, P' \vdash Q}{\Gamma, P \vee P' \vdash Q} \quad \frac{}{\Gamma, \bot \vdash Q}$$

$$\frac{\Gamma \vdash Px}{\Gamma \vdash \forall x.\, Px} \quad \frac{\Gamma, Px \vdash Q}{\Gamma, \exists x.\, Px \vdash Q} \quad \frac{\{(\Gamma \vdash P)\theta : \theta \in csu(u \doteq v)\}}{\Gamma, u = v \vdash P}$$

$$\frac{\Gamma, S\vec{t} \vdash P \quad BS\vec{x} \vdash S\vec{x}}{\Gamma, \mu B\vec{t} \vdash P} \quad \frac{\Gamma, (\mu B\vec{t})^* \vdash P}{\Gamma, \mu B\vec{t} \vdash P}$$

$$\frac{\Gamma \vdash S\vec{t} \quad S\vec{x} \vdash BS\vec{x}}{\Gamma \vdash \nu B\vec{t}} \quad \frac{\Gamma \vdash (\nu B\vec{t})^*}{\Gamma \vdash \nu B\vec{t}}$$

Switching, when no asynchronous rule applies
(P synchronous, Q asynchronous)

$$\frac{\Gamma, Q \Downarrow Q \vdash P}{\Gamma, Q \vdash P} \quad \frac{\Gamma \vdash\Downarrow P}{\Gamma \vdash P}$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \Downarrow P \vdash Q} \quad \frac{\Gamma \vdash Q}{\Gamma \vdash\Downarrow Q}$$

Synchronous phase

$$\frac{}{\Gamma \vdash\Downarrow \top} \quad \frac{\Gamma \vdash\Downarrow P \quad \Gamma \vdash\Downarrow P'}{\Gamma \vdash\Downarrow P \otimes P'} \quad \frac{\Gamma \vdash\Downarrow P_i}{\Gamma \vdash\Downarrow P_0 \vee P_1}$$

$$\frac{\Gamma \Downarrow Pt \vdash Q}{\Gamma \Downarrow \forall x.\, Px \vdash Q} \quad \frac{\Gamma \vdash\Downarrow Pt}{\Gamma \vdash\Downarrow \exists x.\, Px} \quad \frac{}{\Gamma \vdash\Downarrow u = u}$$

$$\frac{\Gamma \Downarrow B(\nu B)\vec{t} \vdash P}{\Gamma \Downarrow \nu B\vec{t} \vdash P} \quad \frac{}{\Gamma \Downarrow \nu B\vec{t} \vdash (\nu B\vec{t})^*}$$

$$\frac{\Gamma \vdash\Downarrow B(\mu B)\vec{t}}{\Gamma \vdash\Downarrow \mu B\vec{t}} \quad \frac{}{\Gamma, (\mu B\vec{t})^* \vdash\Downarrow \mu B\vec{t}}$$

Figure 4.6: The μ-focusing system for μLJ

Chapter 5

Proof-theory and model-checking

The purpose of model-checking is to mechanically verify whether a model satisfies a given property. In practice, models are abstractions of software or electronic systems. Properties are expressed in various logics, often temporal. The class of considered models and specification logics is of course limited by the feasability of automated verification. Since very large systems are usually targetted, simple models are predominent. The typical approach is to encode both the considered system and its specification as finite automata, which reduces model-checking to an automata inclusion: all traces of the system should be accepted by the specification. In this chapter, we shall study how to represent finite automata in μMALL and reason about automata inclusions in that framework.

There are various reasons for considering finite automata in logic, and more precisely proof-theory. First, although most of the effort in the model-checking community is spent on finding more efficient techniques and algorithms, there is a growing interest for producing certificates: when an algorithm returns, it should not only say whether or not a model satisfies a property, but also provide a certificate of that claim. Proof-theory is a natural candidate for that task. Moreover, the ability to express the result of model-checking in a logic makes it possible to integrate it with related tasks like the abstraction of systems and specifications into automata: if the abstractions and the model-checking are proved, cut-elimination allows to conclude that the original system satisfies its specification. Proof-theory offers a common framework for representing, computing and reasoning, mechanically or not.

There is also a proof-theoretical interest in studying finite automata. They form a natural class of complex, infinite behaviors. It is thus a good test for the logic to study how directly an automata can be represented, and how richly the logic allows to reason about it. We shall see that fixed points offer a very natural solution in μMALL. Finite automata are especially interesting in that they are not *too* complex, and have an elegant and well understood theory. In particular, inclusion of finite automata is decidable. This raises two proof-theoretical questions: Is the logic expressive enough to capture all inclusions? Can we carry decidability results to the logic, beyond the particular case of finite automata?

The rest of the chapter is organized as follows. In Section 5.1, we exhibit some

structure to the inclusion property, introducing the concept of multi-simulation, then we propose an encoding of automata as fixed point predicates and obtain soudness and completeness of μMALL for automata inclusions. Then, we apply these observations directly to the logic in Section 5.2, designing and studying the fragment of *regular* formulas. We obtain soundness and completeness for implications between compatible regular predicates, and lay the foundations for new automated theorem proving techniques.

In the rest of the chapter, we work only with first-order terms, so that we have a decidable finitely-branching left equality rule. We shall work in μMALL, usually two-sided, which can safely be read as intuitionistic logic.

5.1 Finite state automata

Definition 5.1 (Finite state automaton, acceptance, language)**.** A non-deterministic finite state automaton on the alphabet Σ is specified by the tuple (Q, T, I, F) where Q is a set whose elements are called *states*, $T \in \wp(Q \times \Sigma \times Q)$ is a set of *transitions* and I and F are subsets of Q, respectively containing the initial and final states. The automaton is said to *accept* a word $c_1 \dots c_n$ when there is a path $q_0 q_1 \dots q_n$ where $q_0 \in I$, $q_n \in F$ and each $(q_{i-1}, c_i, q_i) \in T$. The *language* $\mathcal{L}(A)$ associated to an automaton A is the set of all the words that it accepts.

Definition 5.2 (Acceptance, language for sets of states)**.** Let (Q, T, I, F) be a finite automaton. We say that a set of states $Q' \subseteq Q$ accepts a word $c_1 \dots c_n$ when there is a path $q_0 q_1 \dots q_n$ where $q_0 \in Q'$, $q_n \in F$ and each $(q_{i-1}, c_i, q_i) \in T$. Accordingly, we denote by $\mathcal{L}(Q')$ the set of all the words that Q' accepts. We write $\mathcal{L}(q)$ as shorthand for $\mathcal{L}(\{q\})$.

In particular, the language accepted by an automaton is the language accepted by its initial states, and we have $\cup_i \mathcal{L}(Q_i) = \mathcal{L}(\cup_i Q_i)$.

Definition 5.3 ($\alpha^{-1}L$, $\alpha^{-1}Q$)**.** For a language L and a set of states Q, we define α^{-1}:

$$\alpha^{-1}L \stackrel{def}{=} \{w : \alpha w \in L\} \qquad \alpha^{-1}Q \stackrel{def}{=} \{q' : q \to^{\alpha} q' \text{ for some } q \in Q\}$$

For example, one can check that $\alpha^{-1}\mathcal{L}(Q) = \mathcal{L}(\alpha^{-1}Q)$. We finally come to the only non-standard definition, that is the extension of the notion of transition to sets of states.

Definition 5.4 (Notation for transitions, transitions between collections of states)**.** We write $q \to^{c} q'$ for $(q, c, q') \in T$. For a collection of states Q we write $Q \to^{\alpha} Q'$ when $Q' = \bigcup_{q \in Q} Q_q$ where each Q_q contains only (but not necessarily all) states q' such that $q \to^{\alpha} q'$. In other words, $Q' \subseteq \alpha^{-1}Q$.

In the following we shall not specify on which alphabet each automaton works, supposing that they all work on the same implicit Σ. We also talk about transitions, final and initial states without making explicit the automaton that defines them, since it shall be recovered without ambiguity from the states[1].

[1] States are mere identifiers, and automata are in fact considered modulo renaming. Hence, when several automata are considered we implicitly make the assumption that their sets of states are disjoints (a sort of Barendregt convention), which indeed makes it possible to recover the automaton from one of its states.

5.1.1 Multi-simulation

We propose a coinductive characterization of inclusion. Its interest is to allow the transition from the (semantic) automata-theoretic inclusion to the (syntactic) inductive proof of implication in μMALL. As far as we know, that characterization is as novel as its purpose.

Definition 5.5 (Multi-simulation)**.** A multi-simulation between two automata (A, T, I, F) and (B, T', I', F') is a relation $\mathfrak{R} \subseteq A \times \wp(B)$ such that whenever $p \mathfrak{R} Q$:

- if p is final, then there must be a final state in Q;
- for any α and p' such that $p \to^{\alpha} p'$ there exists Q' such that $Q \to^{\alpha} Q'$ and $p' \mathfrak{R} Q'$.

Proposition 5.6. *$\mathcal{L}(p) \subseteq \mathcal{L}(Q)$ if and only if $p \mathfrak{R} Q$ for some multi-simulation $\mathfrak{R}$.*

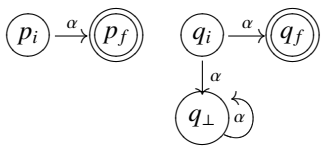
Proof. **Direction "if".** We show that language inclusion is a multi-simulation, which follows immediately from the definition: if we have $\mathcal{L}(p) \subseteq \mathcal{L}(Q)$ and p is final then $\epsilon \in \mathcal{L}(Q)$, hence one of the states of Q must be final; if $p \to^{\alpha} p'$ then $\alpha\mathcal{L}(p') \subseteq \mathcal{L}(Q)$, that is $\mathcal{L}(p') \subseteq \alpha^{-1}\mathcal{L}(Q)$, and hence $Q' := \alpha^{-1}Q$ fits.

Direction "only if". Let $\mathfrak{R}$ be a multi-simulation. We prove by induction on the word w that whenever $p \mathfrak{R} Q$ and $w \in \mathcal{L}(p)$, then $w \in \mathcal{L}(Q)$. It is true for ϵ by definition, as there must be a final state in Q when p is final. If $w = \alpha w'$ is in $\mathcal{L}(p)$ then we have some p' such that $p \to^{\alpha} p'$, and by definition of pre-inclusion there exists Q' such that $Q \to^{\alpha} Q'$ and $p' \mathfrak{R} Q'$. Since $w' \in \mathcal{L}(p')$ we obtain by induction hypothesis that $w' \in \mathcal{L}(Q')$ and hence $w \in \mathcal{L}(Q)$. □

As a side note, the notion of multi-simulation extends naturally to labeled transition systems. In that case, both finite and infinite trace inclusions would be multi-simulations. However, multi-simulation does not imply infinite trace inclusion but only the finite one — which is shown by induction on the length of the trace. As a counter-example, consider on one side a unique state q_ω with a looping transition α, and on the other side all states q_n for $n \in \mathbb{N}$ such that q_{n+1} has an α transition to q_n. There is a multi-simulation between the two, which needs not relate any other pair: q_ω only does an α transition to itself, and so does $\{q_n : n \in \mathbb{N}\}$. The finite traces of ω are $\{\alpha^n : n \in \mathbb{N}\}$, each α^n being doable by state q_n. But the infinite trace α^ω cannot be obtained by any q_n.

We now show a couple examples of how to "prove" some inclusions by means of the multi-simulation technique.

Example 5.7. Inclusion is the greatest multi-simulation, and as such does not give very precise information. Consider for example the following two automata:



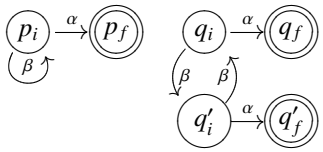
The inclusion relation between states of the left automaton and sets of states of the right one is:

$$\{(p_i, \{q_i\}), (p_f, \{q_f\}), (p_i, \{q_i, q_\perp\}), (p_f, \{q_f, q_\perp\})\}$$

To understand *why* a given p is included in Q, it is more interesting to look at a minimal multi-simulation that relates them. That is what is done for the following examples. In the first direction of the proof of Proposition 5.6 we use $Q' := \alpha^{-1}Q$, which is very imprecise: on our example, it justifies the association $(p_i, \{q_i\})$ by $(p_f, \{q_f, q_\perp\})$. It would have sufficed to take $(p_f, \{q_f\})$. Both alternatives are justified by the finality of q_f, another hint at the uselessness of $q_\perp$ in the first association.

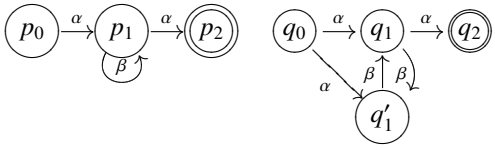
A multi-simulation establishing $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ can be obtained by iterating the following transformation on the relation $\{(p_0, Q_0)\}$: for each (p, Q) and each $p \to^\alpha p'$, add $(p', \alpha^{-1}Q)$ to the relation. When a fixed point is reached, check the condition on final states: if it holds the relation is a multi-simulation; if it does not there cannot be any multi-simulation relating p_0 and Q_0. This simple technique generally gives a smaller relation than inclusion, but it is still not the best: on our example, it yields $\{(p_i, \{q_i\}), (p_f, \{q_f, q_\perp\})\}$.

Example 5.8. Consider the following automata:



The states p_i and q_i accept the same language, namely $\beta^*\alpha$. This is justified by the multi-simulation $\mathfrak{R} = \{(p_i, \{q_i\}), (p_i, \{q'_i\}), (p_f, \{q_f\}), (p_f, \{q'_f\})\}$.

Example 5.9.



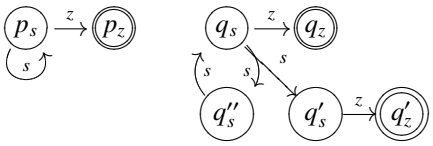
Again, p_0 is included in q_0: if there is an even number of β transitions to make, go to q_1, otherwise go to q'_1. This non-local[2] reasoning is expressed by the multi-simulation $\mathfrak{R} = \{(p_0, \{q_0\}), (p_1, \{q_1, q'_1\}), (p_2, \{q_2\})\}$.

Example 5.10. We finally show an example that hints at the upcoming generalization of this discussion. Consider the inductive specifications of natural numbers and division by two:

$$\begin{array}{lcl} \mathit{nat}\ 0 & \triangleq & \mathbf{1} \\ \mathit{nat}\ (s X) & \triangleq & \mathit{nat}\ X \end{array} \qquad \begin{array}{lcl} \mathit{half}\ 0\ 0 & \triangleq & \mathbf{1} \\ \mathit{half}\ (s 0)\ 0 & \triangleq & \mathbf{1} \\ \mathit{half}\ (s^2 X)\ (s Y) & \triangleq & \mathit{half}\ X\ Y \end{array}$$

[2] One can sense here the local but backwards aspect of proofs by induction.

Our goal here is to establish the totality of $half$, that is $\forall x.\, nat\ x \multimap \exists y.\, half\ x\ y$. We do so by first extracting the finite automata describing the behaviors of these specifications, and then checking their inclusion. The first step is informal, which will be fixed in Example 5.21. The transformation is easy for nat; for $half$ it requires to erase the information about the second parameter:



The following multi-simulation establishes that $\mathcal{L}(p_s) \subseteq \mathcal{L}(q_s)$:

$$\mathfrak{R} = \{(p_s, \{q_s\}), (p_s, \{q'_s, q''_s\}), (p_z, \{q_z\}), (p_z, \{q'_z\})\}$$

We have exhibited a simple structure underlying inclusion of non-deterministic finite automata, which expresses non-trivial reasoning. We are now going to move to the (linear) logical world and exploit it.

5.1.2 Encoding finite automata in μMALL

We shall represent an automaton, or rather its acceptance predicate, in the logic μMALL. A natural first step is to translate the automaton into a set of definitions, with one predicate per state, one clause per transition and one clause per final state:

$$\begin{array}{lcll} q(\alpha w) & \stackrel{\triangle}{=} & q'w & \text{for each } q \to^{\alpha} q' \\ q\epsilon & \stackrel{\triangle}{=} & \mathbf{1} & \text{for each final state } q \end{array}$$

This yields an exact representation of acceptance for a given automaton: there is a bijection between the accepting paths starting with one of its states q and proofs[3] of the corresponding predicate. Note that the information of the names of states is kept, and as a result the representation does not identify automata that differ only by a renaming of their states. A more serious drawback of definitions, for the task that we consider, is that it is very cumbersome to deal with mutually inductive definitions. Indeed, the only published work on induction for arbitrary definitions [MT03b, Tiu04] requires stratification, which rules out mutually inductive definitions.

These definitions can then be refactored to have only one clause per predicate and flexible arguments in the head, without changing the properties of the representation. The resulting clauses would have the following form:

$$qw \stackrel{\triangle}{=} w = \epsilon \oplus (\exists w'.\, w = \alpha w' \otimes q'w') \oplus \ldots$$

At this point, two alternatives should be considered. This definition can be turned into a single fixed point by keeping the names of the states, passing them as an extra

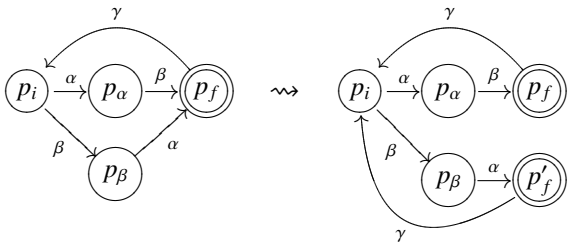
[3]These proofs can be in any logic supporting definitions, this detail does not matter, especially since we are considering extremely simple formulas.

argument:

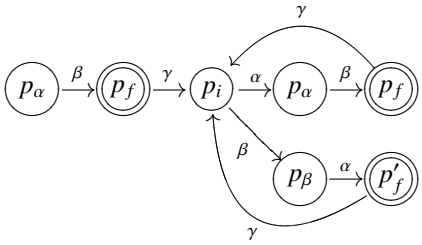
$$accept \stackrel{def}{=} \mu(\lambda A \lambda x \lambda w. \quad x = q \otimes (w = \epsilon \oplus (\exists w'.\ w = \alpha w' \otimes A\ q'\ w') \oplus \ldots) \oplus x = q' \otimes \ldots)$$

We are interested in a more *structural* approach, that erases the names of states and translates the mutually inductive definitions into a interleaved fixed points. Moreover, this shall bring nice examples of how simultaneous induction schemes arise naturally from the simple induction rule.

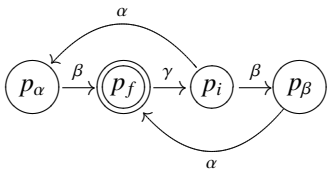
The drawback of interleaved fixed points, compared to definition tables, is that it forces a sequential introduction of mutually inductive predicates, which forbids sharing. Before studying formally this issue, we shall give an intuition of it on automata. Graphically, that sequentialization requires writing a graph as a tree (the syntax tree) plus looping edges (predicate variables referring to fixed points). For example, the following transformation will essentially be applied when encoding p_i as a μ-formula:



Taking the α transition from p_i involves an unfolding of the μ formula:



On the other hand, the encoding of the state p_α of the original automaton will not exactly be the same. It is obtained by taking a covering tree rooted at p_α in the automaton:



Notice, however, that these mismatches preserve bisimulation.

We shall now describe formally how mutual inductive definitions can be translated as interleaved fixed points. This is of course a more general question than just encoding finite automata.

Definition 5.11 (Translation of defined atoms into fixed points)**.** Let $(a\vec{x} \triangleq B_a\vec{x})_a$ be a collection of mutually inductive definitions, in the form where there is a single clause per defined atom and the head variables are flexible. We translate the defined atoms into fixed point expressions, generally encoding a formula F into $[F]$ (shorthand for $[F]^{\cdot}$) as follows:

$$\begin{array}{rcl}
[a\vec{t}]^{\Gamma} & \equiv & \begin{cases} \mu(\lambda p.\lambda\vec{x}.\ [B_a\vec{x}]^{\Gamma,\, a\mapsto p})\vec{t} & \text{if } a \text{ is unbound in } \Gamma \\ p\vec{t} & \text{if } \Gamma \text{ binds } a \text{ to } p \end{cases} \\
[u = v]^{\Gamma} & \equiv & u = v \\
[P * Q]^{\Gamma} & \equiv & [P]^{\Gamma} * [Q]^{\Gamma} \\
[Qx.\ Px]^{\Gamma} & \equiv & Qx.\ [Px]^{\Gamma}
\end{array}$$

Here, $*$ stands for any propositional binary connective and Q for any quantifier.

Notice that this encoding is structural: it does not rely on the names of the defined atoms but only reflects their structure. As we have shown in the previous examples, bisimilar states might be identified.

Example 5.12. Let us develop the syntactic counterpart of the graphical explanation above. We shall translate into a fixed point the atom p_i defined by the following table:

$$\begin{array}{rcl}
p_i w & \triangleq & (\exists w'.\ w = \alpha w' \otimes p_\alpha w') \oplus (\exists w'.\ w = \beta w' \otimes p_\beta w') \\
p_\alpha w & \triangleq & (\exists w'.\ w = \beta w' \otimes p_f w') \\
p_\beta w & \triangleq & (\exists w'.\ w = \alpha w' \otimes p_f w') \\
p_f w & \triangleq & (w = \epsilon) \oplus (\exists w'.\ w = \gamma w' \otimes p_i w')
\end{array}$$

The translation of p_i starts with the empty context: $[p_i] \equiv [p_i]^{\cdot} \equiv \lambda w.\ [p_i w]^{\cdot}$. Since p_i is unbound in the empty context, the translation introduces a least fixed point, associating it to p_i in the context, and unfolds p_i into its definition:

$$[p_i w]^{\cdot} \equiv \mu(\lambda p_i'\lambda w.\ [(\exists w'.\ w = \alpha w' \otimes p_\alpha w') \oplus (\exists w'.\ w = \beta w' \otimes p_\beta w')]^{p_i\mapsto p_i'})w$$

Then, the translation commutes with first-order and propositional connectives, to reach the defined atoms p_α and p_β:

$$[p_i w]^{\cdot} \equiv \mu(\lambda p_i'\lambda w.\ (\exists w'.\ w = \alpha w' \otimes [p_\alpha w']^{p_i\mapsto p_i'}) \oplus (\exists w'.\ w = \beta w' \otimes [p_\beta w']^{p_i\mapsto p_i'}))w$$

These two atoms are treated in the same way: since they are unbound in the associated contexts, a new least fixed point based on the translation of their definitions is introduced. After trivially pushing the translation down to p_f, we obtain the following:

$$\begin{array}{ll}
\mu(\lambda p_i'\lambda w. & (\exists w'.\ w = \alpha w' \otimes \mu(\lambda p_\alpha'\lambda w.\ \exists w'.\ w = \beta w' \otimes [p_f w']^{p_\alpha\mapsto p_\alpha', p_i\mapsto p_i'})w') \\
& \oplus (\exists w'.\ w = \beta w' \otimes \mu(\lambda p_\beta'\lambda w.\ \exists w'.\ w = \alpha w' \otimes [p_f w']^{p_\beta\mapsto p_\beta', p_i\mapsto p_i'})w'))w
\end{array}$$

Then, the definition of p_f is unfolded:

$$\begin{aligned}
&\mu(\lambda p_i'\lambda w.\\
&\quad(\exists w'.\ w = \alpha w' \otimes \mu(\lambda p_\alpha'\lambda w.\ \exists w'.\ w = \beta w' \otimes\\
&\qquad\mu(\lambda p_f'\lambda w.\ (w = \epsilon) \oplus (\exists w'.\ w = \gamma w' \otimes [p_i w']^{p_f \mapsto p_f', p_\alpha \mapsto p_\alpha', p_i \mapsto p_i'}))w')w')\\
&\quad\oplus(\exists w'.\ w = \beta w' \otimes \mu(\lambda p_\beta'\lambda w.\ \exists w'.\ w = \alpha w' \otimes\\
&\qquad\mu(\lambda p_f'\lambda w.\ (w = \epsilon) \oplus (\exists w'.\ w = \gamma w' \otimes [p_i w']^{p_f \mapsto p_f', p_\beta \mapsto p_\beta', p_i \mapsto p_i'}))w')w'))w
\end{aligned}$$

We finally cycle, from p_f back to p_i. This time, p_i is bound in the context so it is translated to the associated predicate variable:

$$\begin{aligned}
[p_i w]^{\cdot} \equiv \mu(\lambda p_i'\lambda w.&\\
&(\exists w'.\ w = \alpha w' \otimes \mu(\lambda p_\alpha'\lambda w.\ \exists w'.\ w = \beta w' \otimes\\
&\quad\mu(\lambda p_f'\lambda w.\ (w = \epsilon) \oplus (\exists w'.\ w = \gamma w' \otimes p_i' w'))w')w')\\
&\oplus(\exists w'.\ w = \beta w' \otimes \mu(\lambda p_\beta'\lambda w.\ \exists w'.\ w = \alpha w' \otimes\\
&\quad\mu(\lambda p_f'\lambda w.\ (w = \epsilon) \oplus (\exists w'.\ w = \gamma w' \otimes p_i' w'))w')w'))w
\end{aligned}$$

The resulting formula does correspond to the above expansion of our automata, written as a tree with looping edges, which involves a copy of p_f.

Proposition 5.13 (Adequacy of the structural encoding)**.** *Let $\mathcal{A}$ be a finite automaton. There is a bijection between accepting paths starting at one of its state q and μMALL derivations of $\vdash [qw]$, where w is the word induced by the path.*

Proof. We construct an injective mapping from paths to derivations, by induction on the length of the path. Along the same lines, by induction on the derivation, we would build a mapping from derivations to paths. Together, this establishes the existence of a bijection — without resorting to this general argument, one can also simply observe that the two mappings are inverse of each other. The difficulty is that the encoding of a state depends on Γ, *i.e.,* on how it has been reached in the graph. But we only need to consider *valid* contexts: a context is said to be valid if it is empty, or if it of the form $(\Gamma, a \mapsto [a]^\Gamma)$ for a valid Γ that does not contain a.

We establish the following generalization: for each valid Γ and each state q, there is an injective mapping from accepting paths starting at q and derivations of $\vdash [qw]^\Gamma$, where w is the word associated to the path. These mappings are all defined simultaneously, by induction on the path and sub-induction on the validity of the context:

- If $q \in \Gamma$ then $[q]^\Gamma$ is $\Gamma(q)$, which is of the form $[q]^{\Gamma'}$ for a predecessor of Γ. We can thus conclude by the sub-induction hypothesis.
- Otherwise $[q]$ is a fixed point obtained by encoding B_q:
  - If the path is empty, q is accepting and w is ϵ. Then $\vdash [qw]^\Gamma$ is derived by unfolding and selecting the (only) disjunct $w = \epsilon$ in B_q.
  - If the path starts with $q \to^\alpha q'$, the derivation is obtained by unfolding, selecting the (only) disjunct corresponding to the transition, and completed by inductive hypothesis, on q' and $(\Gamma, q \mapsto [q]^\Gamma)$.

□

The only interest of that proof is the formalization of the valid contexts, since the local dynamics matched in a straightforward way. Although it is actually not necessary to the proof, it shows that there are only a finite number of contexts that can occur during the superficial[4] unfoldings of an encoding. In other words, it formalizes what is very intuitive on graphs, or rather trees with looping edges: there are only a finite number of graphs (fixed points) that arise from all possible traversals (superficial unfoldings).

Remark 5.14. Having outlined a general translation from definitions to fixed points, it is natural to wonder what kind of result can we obtain about it. As long as no implications are used in the body of the definitions, the previous proof can be extended to obtain adequacy between derivations of $\vdash \vec{at}$ in a logic with definitions and derivations of $\vdash [\vec{at}]$ in the corresponding logic equipped with fixed points.

However, it is hard to go further than that. As soon as negations occur, the derivation of the encoding may use induction rules, possibly with invariants that are not encodings of formulas using defined atoms. More importantly, the derivation of the encoding can use axiom rules which cannot be translated to axioms on the other side, because two distinct atoms identically defined are equalized by the encoding. In order to recover the missing identities in presence of mutually inductive definitions, one needs mutual induction schemes, which has not been so much studied yet.

Anyway, it is not so important to relate these two systems; let us rephrase the question. Consider a system working with least and greatest fixed points, where the user would type in specifications in the more convenient style of definitions. What can we guarantee to the user about our translation of his input? This question is not a trivial one: we have seen that some fixed points might occur at toplevel which are not subformulas of the user input; going further, there are surprising examples when mixing inductive and coinductive definitions.

Example 5.15. When the definitions are all inductive or all coinductive, on can check that $[a\vec{x}] \multimap [B_a\vec{x}]$, but this is not necessarily the case for definitions interleaving least and greatest fixed points. Consider the natural extension of our translation to tables mixing inductive and coinductive definitions: clauses would be annotated with μ and ν, and this annotation would be used when translating a defined atom into a fixed point. Consider the following definitions: $p \stackrel{\triangle}{=}_\mu q$ and $q \stackrel{\triangle}{=}_\nu p$. Then $[p] \equiv \mu p.\ \nu q.\ p$ which is equivalent to $\mu p.\ p$ and hence to $\mathbf{0}$. And $[q] \equiv [p]^\perp$, equivalent to $\top$.

We leave to future work the exploration of these issues and the design of solutions, if needed.

5.1.3 Completeness

We have adequately represented automata in μMALL, and shall now consider reasoning about them in our logic. We are going to exploit (a small amount of) the μ-focusing

[4] Obviously, infinitely many subformulas result from the unfolding of fixed points deep inside a formula. But that is not what occurs in a cut-free derivation.

of μMALL, obtaining a synthetic reading of the positive fixed points that represent automata.

Proposition 5.16. *Let A be a finite automaton, and p one of its states. The following rule is sound and invertible, where the states p', p'' are taken among those reachable from p:*

$$\frac{\{\vdash S_{p'}\epsilon : p' \textit{ final}\} \quad \{S_{p''}x \vdash S_{p'}(\alpha x) : p' \to^{\alpha} p''\} \quad S_0\vec{t} \vdash \Gamma}{[p]\vec{t} \vdash \Gamma}$$

Like the induction rule, this rule leaves the difficult choice of finding a correct invariant for each state.

Proof. Let μA_p be the encoding $[p]$. We observe that $(\mu A_p)^{\perp}$ is fully asynchronous, and can hence be fully decomposed in an invertible way. In the focusing system, freezing would have to be considered but it can be ignored here since we are only interested in invertibility — this comes at the price of possible unnecessary unfoldings. The rule of the decomposition is an induction:

$$\frac{A_p S_p x \vdash S_p x \quad S_p t \vdash \Gamma}{\mu A_p t \vdash \Gamma}$$

By applying more asynchronous rules on A_p, we enumerate all transitions of p, and the terminal case if it is final:

$$\frac{\vdash S_p\epsilon \text{ if } p \text{ is final} \quad \{[p']^{p\mapsto S_p}x \vdash S_p(\alpha x) : p \to^{\alpha} p'\} \quad S_p t \vdash \Gamma}{\mu A_p t \vdash \Gamma}$$

Following that scheme, induction and case analysis are added, parsing the whole covering tree rooted at p. For example, the main premise of the induction on $[p']^{p\mapsto S_p}$ above, whose invariant shall be called $S_{p'}$, will be $S_{p'}x \vdash S_p(\alpha x)$. As the tree is parsed, each transition becomes an implication between the corresponding invariants. Finally, some transitions might go back in that tree: in the fixed points they will be represented by a call to an outer inductive predicate, which is substituted in our case by the corresponding invariant. For example if $p' \to^{\beta} p$, then after the induction on $[p']^{p\mapsto S_p}$ and the subsequent case analysis, one of the premises will be $S_p x \vdash S_{p'}(\alpha x)$.

We have seen that one state of an automaton is not uniquely represented in the fixed point encoding, but might occur in different forms depending on how it was reached in the graph. This applies to the parsing used to derive the simultaneous induction rule, but since all variations impose the same constraints on their invariants, we can factorize the corresponding premises. □

We have seen that the mutual induction rule emerges naturally from the simple one, especially in the case of a fully asynchronous definition since all the steps can be considered as one big step without sacrificing completeness. We shall now use that macro-rule for proving inclusions.

Theorem 5.17. *Let A and B be two automata, let p_0 be a state of A and Q_0 a collection of states of B. Then $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ if and only if $\forall x.\, \mu A_{p_0} x \multimap \oplus_{q\in Q_0} \mu B_q x$ is provable in μMALL.*

Proof. The easy direction here is to conclude the inclusion from the provability of the linear implication. We use the adequacy of the encoding: whenever $w \in \mathcal{L}(p_0)$ we get $\vdash \mu A_{p_0} w$, which we cut against the implication proof to obtain $\vdash \oplus_{q \in Q_0} \mu B_q w$. After eliminating the cut from that derivation, we only have to inspect it: after the $\oplus$ we have a derivation of $\vdash \mu A_q w$ for some $q \in Q$, which implies that q accepts w.

For the other direction we use our characterization of inclusion. Since $\mathcal{L}(p_0) \subseteq \mathcal{L}(Q_0)$ there exists a multi-simulation $\mathfrak{R}$ that relates them. We prove $\mu A_{p_0} x \vdash \oplus \mu B_q x$ by using the simultaneous induction rule shown above, with the invariants S_p given as follows by the multi-simulation:

$$S_p := \lambda x.\ \&_{p \mathfrak{R} Q} \oplus_{q \in Q} \mu B_q x$$

We have to build a proof of $\vdash S_p \epsilon$ for each terminal state p: we enumerate all $p \mathfrak{R} Q$ by introducing the &, then since $\mathfrak{R}$ is a multi-simulation there must be a final state $q_f \in Q$, we select this disjunct and finally derive $\vdash \mu B_{q_f} \epsilon$ by selecting the final clause in it.

We also have to build a derivation of $S_{p'} x \vdash S_p(\alpha x)$ for each $p \to^\alpha p'$. We introduce again the & on the right, enumerating all $p \mathfrak{R} Q$, then by definition of the multi-simulation there is a Q' such that $Q \to^\alpha Q'$ and $p' \mathfrak{R} Q'$, so we choose the corresponding &-conjunct on the left hand-side. We are left with $\oplus_{q' \in Q'} \mu B_{q'} x \vdash \oplus_{q \in Q} \mu B_q(\alpha x)$ which corresponds exactly to $Q \to^\alpha Q'$: for each q' there is a q such that $q \to^\alpha q'$. We translate that by enumerating all q' (introducing the left $\oplus$) and choosing the appropriate q in each branch (introducing the right $\oplus$) and finally selecting the right clause in B_q to establish $\mu B_{q'} x \vdash \mu B_q(\alpha x)$. □

This theorem shows once more that μMALL is an expressive framework, which is sufficient (and probably necessary) for reasoning on automata inclusion. Our representation of automata as fixed points is very satisfying: the derived induction principle allows rich reasoning about acceptances, naturally reflecting the behavior of an automaton. It fits perfectly with the notion of multi-simulation. The natural next step is to try to obtain a similar logical support for more complex automata, for example, Büchi automata. We shall briefly discuss it, before putting automata theory in the background and applying the same methodology directly on logic to obtain a more general result.

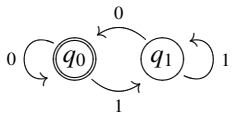
5.1.4 Büchi automata

A very interesting thing to consider is Büchi automata, but it is also very challenging. Büchi automata are much more complex than finite automata. For example, non-deterministic ones cannot necessarily be determinized, which has motivated an active research on algorithms working directly on non-deterministic automata. Reasoning on infinite traces is very complex, which makes it especially appealing to use logic in order to avoid errors and generally guide our understanding of the involved structures.

The first obstacle to work with infinite automata and hence infinite words in μMALL is to add infinite terms. It would affect equality, since the occur-check rule becomes invalid. Another problem with Büchi automata is the adequacy result, because acceptance is defined coinductively. This is an essential problem that cannot be

avoided: when coinduction is involved, one has to make some assumptions on the language in which the coinvariant is represented in order to obtain its representation in the logic. Even if the ultimate goal of adequacy is out of reach, some care should be taken to understand better the gap between defined atoms and fixed points when inductive and coinductive definitions are interleaved (see Example 5.15).

There are a lot of challenges, but also some promising examples. Consider the following Büchi automaton:



Both states accept words that pass infinitely many times through the accepting state q_0, that is words containing infinitely many 0.

To encode it in the logic, we define as coinductive the states that are final, the other being inductive as before. In definition style we obtain:

$$q_0 x \stackrel{\triangle}{=}_\nu (\exists w.\ x = 0w \otimes q_0 w) \oplus (\exists w.\ x = 1w \otimes q_1 w)$$

$$q_1 x \stackrel{\triangle}{=}_\mu (\exists w.\ x = 0w \otimes q_0 w) \oplus (\exists w.\ x = 1w \otimes q_1 w)$$

We obtain the following translation for q_0:

$$q_0 \equiv \nu q_0 \lambda x.\ (\exists w.\ x = 0w \otimes q_0 w) \oplus (\exists w.\ x = 1w \otimes (\mu q_1 \lambda x.\ (\exists w.\ x = 0w \otimes q_0 w) \oplus (\exists w.\ x = 1w \otimes q_1 w))w)$$

In this hypothetical extension of μMALL with infinite terms, we can also naturally specify that a word is an infinite string of 1 (resp. 0); we call these greatest fixed points inf_1 (resp. inf_0).

Our encoding of the automaton turns out to behave interestingly well on some example properties. We first consider proving that q_0 accepts the infinite chain of 0:

$$\forall x.\ inf_0\ x \multimap q_0\ x$$

After having treated $\forall$ and $\multimap$, we are left with the dual of a coinductive, and a coinductive which should then be treated in the asynchronous phase. Indeed, the trivial coinduction works easily.

We also consider a negative result:

$$\forall x.\ q_0\ x \multimap inf_1\ x \multimap \mathbf{0}$$

This time, the first asynchronous phase leaves the duals of the two greatest fixed points untouched. We must focus on one to unfold it, and choose $inf_1\ x$. After a second asynchronous phase we are left with $inf_1\ y, q_0\ (1y) \vdash \mathbf{0}$, and focus on q_0. After its unfolding, the third asynchronous phase does the case analysis on possible transitions, and we obtain $inf_1\ y, q_1\ y \vdash \mathbf{0}$. But the asynchronous phase does not end because we have an inductive on the left, namely q_1: this constitutes a handle on infinite behaviors. The obvious invariant ($\lambda y.\ inf_1\ y \multimap \mathbf{0}$) allows to conclude easily.

5.2 Regular formulas

We obtained a completeness result for finite automata, behind which lies a decision procedure that builds complex invariants. It is tempting to extend such a good property. Obviously, the completeness result obtained for representations of finite automata can be generalized to fixed points of similar shape. In this section, we consider a notion of *regular formulas* that is considerably richer, in an attempt to capture simple but useful properties like totality of relations. As finite automata, regular formulas are finite systems of interdependent superficial constraints. The main differences between finite automata and regular formulas are that our formulas deal with terms rather than words and have an arbitrary arity. Hence, regular formulas do not specify only languages but also relations. We shall see that this possibility makes regular formulas much more complex than finite automata — the name "regular" is thus misleading. While only the first letter of a word is checked when taking a transition in an automata, terms in regular formulas are matched against arbitrary patterns. In particular, the pattern can be trivial, which corresponds to an ϵ-transition.

In this section, we restrict ourselves to first-order terms in order to have a well-behaved unification.

Definition 5.18 (Patterns)**.** A *pattern* C of type $\gamma_1, \ldots, \gamma_n \to \gamma'_1, \ldots, \gamma'_m$ is a vector of m closed terms[5] $p_i : \gamma_1, \ldots, \gamma_n \to \gamma'_i$, such that each of the n variables occurs at most once in all $(p_i)_i$. The (p_i) are called elementary patterns of C. A pattern is said to be *non-erasing* when each of its input variables occurs in one of its elementary patterns.

We write $C\vec{t}$ for denoting the vector resulting from the application of $\vec{t}$ to each elementary pattern of C. For two vectors of terms of equal length n, $\vec{t} = \vec{t'}$ denotes the formula $t_1 = t'_1 \otimes \ldots \otimes t_n = t'_n$.

An elementary pattern can be seen as a tree with term constants as nodes and either term constants or variables as leafs. Using this viewpoint, we define the size of a pattern to be the total number of nodes of its elementary patterns, and its height to be the maximum of their heights.

A *trivial pattern* is a pattern which has no rigid structure, *i.e.,* whose elementary patterns are projections. Trivial patterns are denoted by ϵ. A particular case of trivial pattern is the identity: we denote by $\mathcal{I}_n$ the pattern $(\lambda\vec{x}.x_1, \ldots, \lambda\vec{x}.x_n)$.

Definition 5.19 (Pattern compositions)**.** Let C and C' be patterns of arbitrary type. Let $(p_i)_{i\leq m}$ be the elementary patterns of C and $(p'_j)_{j\leq m'}$ those of C'. We define (C, C') to be $(\lambda\vec{x}\vec{y}.p_1\vec{x}, \ldots, \lambda\vec{x}\vec{y}.p_m\vec{x}, \lambda\vec{x}\vec{y}.p'_1\vec{y}, \ldots, \lambda\vec{x}\vec{y}.p'_{m'}\vec{y})$, which is still a pattern.

Assuming that C has type $\vec{\gamma} \to \vec{\gamma'}$, and C' has type $\vec{\gamma'} \to \vec{\gamma''}$, we define $C'C$ to be the pattern $(\lambda\vec{x}.p'_1(C\vec{x}), \ldots, \lambda\vec{x}.p'_{m'}(C\vec{x}))$.

Definition 5.20 (Regular formula)**.** We define the class of formulas $\mathcal{R}^{\Gamma}_{I/O}$, parametrized by Γ (a set of predicate variables), I (a set of *input* term variables) and O (a set of *output* variables). The *regular formulas* on a signature Σ are given by $\mathcal{R}^{\emptyset}_{\emptyset/\Sigma}$.

$$\mathcal{R}^{\Gamma}_{I/O} \quad ::= \quad \mathcal{R}^{\Gamma}_{I/O} \oplus \mathcal{R}^{\Gamma}_{I/O} \mid \exists y.\, \mathcal{R}^{\Gamma}_{I,y/O} \mid \mathcal{P}^{\Gamma}_{I\cup O}$$

[5] The p_i are not first-order terms but they are only a technical device for presenting patterns. The point is that they shall always be applied when occurring in formulas, hence yielding terms of ground type γ'.

$$
\begin{array}{rcl}
 & | & O = C \text{ when } I = \emptyset \\
 & | & O' = CI \otimes \mathcal{P}^{\Gamma}_{I \cup O''} \text{ when } O' \text{ and } O'' \text{ form a partition of } O \\
\mathcal{P}^{\Gamma}_{I} & ::= & p\vec{x} \mid \mu(\lambda p.\lambda\vec{x}.\, \mathcal{R}^{\Gamma,p}_{\emptyset/\vec{x}})\vec{x} \text{ where } \vec{x} \text{ is } I \text{ in an arbitrary order}
\end{array}
$$

We say that a predicate P is regular when $P\vec{x}$ is regular over the signature $\vec{x}$.

The syntactic definition of regular formulas is quite restrictive but suffices to capture interesting examples. In particular, encodings of finite automata are regular formulas. In the last clause of $\mathcal{R}$, notice that the splitting of O allows that some unconstrained output variables are passed to the recursive occurrence $\mathcal{P}$. This allows direct encodings of ϵ-transitions, without resorting to an artificial clause of the form $\lambda w.\, \exists w'.\, w = w' \otimes p'\ w'$ for copying the input variable to the output.

Notice that the fixed point subformulas of a regular formula do not have free term variables. Hence, regular formulas are encodings of definitions.

Example 5.21. Both $(\lambda x.\, nat\ x)$ and $(\lambda x.\, \exists h.\, half\ x\ h)$ are regular predicates. The usual specification of addition would also be regular, but not that of multiplication.

Example 5.22. It is not possible to encode automata with (unbounded) state, as it would require to pass constructed terms to recursive occurrences. This can be understood by considering the pushdown automata recognizing the words $0^n 1^n$, encoded as follows with its stack as first argument:

$$
\begin{array}{llll}
p_0\ s\ (0w) \stackrel{\triangle}{=} p_0\ (0s)\ w & & p_1\ (0s)\ (1w) \stackrel{\triangle}{=} p_1\ s\ w \\
p_0\ s\ w \stackrel{\triangle}{=} p_1\ s\ w & & p_1\ \epsilon\ \epsilon \stackrel{\triangle}{=} \mathbf{1}
\end{array}
$$

We now exhibit a fundamental property of regular formulas, which shall allow us to abstract away from their syntactic definition.

Proposition 5.23 (Fundamental property)**.** *Let P be a regular predicate. There is a finite collection of (regular) predicates (P_i), called* states *of P, such that:*

- *P_0 is P.*
- *Each $P_i\vec{x}$ is provably equivalent to an additive disjunction of formulas of the form $\exists\vec{y}.\ \vec{x} = C\vec{y}$ or $\exists\vec{y}.\ \vec{x} = C\vec{y} \otimes P_j\vec{y}$:*

  $$\forall\vec{x}.\ \left(P_i\vec{x} \multimap (\exists\vec{y}.\ \vec{x} = C\vec{y}) \oplus (\exists\vec{y}.\ \vec{x} = C'\vec{y} \otimes P_j\vec{y}) \oplus \dots\right)$$

  When the first form occurs in the disjunction we say that P_i is C-final; when the second one occurs we write $P_i \to^{C'} P_j$.
- *The following rule is admissible:*

  $$\frac{\{ \vdash S_i C\ :\ P_i\ C\text{-final} \}\quad \{\ \vec{x}; S_j\vec{x} \vdash S_i(C\vec{x})\ :\ P_i \to^{C} P_j\ \}\quad S_0\vec{t} \vdash \Gamma}{P\vec{t} \vdash \Gamma}$$

Proof. The essential observation is the following. Consider a cut-free derivation of some instance $\vdash P\vec{t}$. After a first unfolding in the case where P is a fixed point, either

the provability of $P\vec{t}$ is reduced to that of some (regular) $\mu B\vec{t'}$, or the proof is completed without any μ rule. In other words, our derivation must have one of the following forms, where we indicate multiple rule applications by regular expressions:

$$\dfrac{\dfrac{}{\vdash \vec{t} = C}\,(\otimes|=)^*}{\vdash P\vec{t}}\,\mu^?\oplus^* \qquad \dfrac{\dfrac{}{\vdash \vec{t}_{O'} = C'\vec{t}_I}\,(\otimes|=)^* \quad \dfrac{\vdots}{\vdash \mu B\vec{t}_{I\cup O''}}}{\vdash P\vec{t}_O}\,\mu^?(\oplus|\exists)^*$$

This simply follows from the definition of regular formulas. In the second case, we indicate in subscript on terms to which variables of the definition they correspond. We observe that the same (partial) derivations are also suitable for deriving $P(C\vec{x})$ for distinct eigenvariables $\vec{x}$. In other words, we can derive the following generic facts, after modifying a little C' into C'' to adapt to the order of variables imposed by μB and so that essentially O'' is empty:

$$\vec{x};\ \vec{x} = C \vdash P\vec{x} \qquad \vec{x};\ \exists\vec{y}.\ \vec{x} = C''\vec{y} \otimes \mu B\vec{y} \vdash P\vec{x}$$

It still holds by forming, on the left, the disjunction of all possibilities, *i.e.,* all final cases and transitions. This gives us one direction of the characterization. Finally, since we have enumerated all possibilities of proving P, the converse also holds:

$$\vec{x};\ P\vec{x} \vdash (\vec{x} = C) \oplus (\exists\vec{y}.\ \vec{x} = C''\vec{y} \otimes \mu B\vec{y}) \oplus \dots$$

It is derived by applying asynchronous rules on P, unfolding it if it is a fixed point, which leads symmetrically to one of the transitions or final cases, at which point it is easy to close the derivation.

We characterized the regular formula P by its final patterns and its transitions to some other regular formulas μB. We can repeat the same process on these new formulas. We claim that this process eventually loops, which yields a finite collection of states (P_i). Indeed, even though the derivations of $P\vec{t}$ might involve complex unfoldings (*cf.* Section 5.1.2), only a limited number of formulas can appear. Consider all fixed point bodies occurring in P: they are of the form $(\lambda(p_i)_i \lambda p \lambda\vec{x}.\ B(p_i)_i p\vec{x})$ where the (p_i) denote the fixed points introduced above and p denotes the current one. Note in particular that each B is closed, which would not be true in general but follows from regularity. In a proof of P, fixed points $\mu B_0\vec{t}$ of depth 0 would arise first, and would be unfolded into $B_0(\mu B_0)\vec{t}$. New formulas cannot arise from the recursive occurrence μB_0, but only from a fixed point of depth one occurring in B_0. It would be of the form $\mu(B_1(\mu B_0))\vec{t}_1$, unfolding into $B_1(\mu B_0)(\mu(B_1(\mu B_0)))\vec{t}_1$. Again, new formulas can only arise from B_1. The same pattern applies as we go deeper — this is similar to the valid contexts of Proposition 5.13. Since the depth is bounded by the syntax tree of P, there are only a finite number of such unfoldings to consider, so only a finite number of distinct μB can occur at toplevel in cut-free derivations.

We finally describe how to obtain the simultaneous induction rule. If P is a fixed point, we can consider using the general induction rule instead of unfolding it on the left hand-side of a sequent. The eager application of induction and other asynchronous rules leads to the simultaneous induction rule in the same way as in Proposition 5.16.

If $P = P_0$ is not a fixed point, then there is no transition that leads to it. We start by cutting S_0, the final premises $\vdash S_0 C$ allow to derive the final branches of $P\vec{t} \vdash S_0\vec{t}$. When the derivation reaches a fixed point $P_i\vec{t}$ (*i.e.,* $P_0 \to^C P_i$) we apply the previous construction on P_i, and $S_i\vec{t} \vdash S_0(C\vec{t})$ appears as the main premise of the induction. □

Corollary 5.24. *The provability of* $\vdash P\vec{t}$*, and more generally of* $\vdash \exists\vec{x}.\ P(C\vec{x})$*, is decidable.*

Proof. This simple observation relies on the fact that only finitely many fixed points can occur during the depth-first proof-search for $P(C\vec{x})$, and that their term parameters can only decrease in size. Hence, it is possible to detect looping branches, which cannot lead to any derivation, and abort them. And the exhaustive search terminates. □

5.2.1 Internal completeness

For proving a regular formula P by induction on an other regular formula Q, we are going to need to adapt the states of P, so that their behavior is finitely defined for the transitions of Q, which might be finer than those of P.

Definition 5.25 (Q-states)**.** Let P and Q be regular predicates of the same type. We say that P admits *Q-states* if there is a finite number of predicates (P'_i) such that:

- P is equivalent to P'_0
- for each transition C of Q, each P'_i of compatible type, $P'_i(C\vec{x})$ is provably equivalent to an additive disjunction of $P'_j\vec{x}$

Theorem 5.26. *Let P and Q be two regular predicates of same type such that P admits Q-states.*

$$\{\ \vec{t} \colon \vdash Q\vec{t}\ \} \subseteq \{\ \vec{t} \colon \vdash P\vec{t}\ \} \quad \textit{if and only if} \quad \vec{x}; Q\vec{x} \vdash P\vec{x}$$

Proof. If we have a derivation of the implication we obtain the inclusion by cut-elimination. For the other direction, we use a technique similar to the first part of the proof of Proposition 5.6.

When P' and Q' are predicates of the same type, we simply write $P' \subseteq Q'$ for $\{\ \vec{t} \colon \vdash P'\vec{t}\ \} \subseteq \{\ \vec{t} \colon \vdash Q'\vec{t}\ \}$. We shall build a derivation that uses the derived induction rule on Q (Proposition 5.23). Consider the Q-states of P, called $(P'_i)_{i\leq n}$. For each state Q_i, we form the conjunction of all unions of Q-states of P that contain Q_i:

$$S_i := \&\{\ \oplus_k P'_{i_k}\ \colon Q_i \subseteq \oplus_k P'_{i_k}\ \}$$

We check that the (S_i) are valid invariants for Q:

- For each C-final Q_i, we have by definition of S_i an acceptance of C in each conjunct, which allows to prove $\vdash S_i C$.
- For each transition $Q_i \to^C Q_j$, we need to derive $S_j\vec{x} \vdash S_i(C\vec{x})$. Our derivation starts by a & rule which enumerates all conjuncts S of S_i. Each S contains Q_i by definition of S_i, and by definition of the Q-states there is another disjunction of Q-state S' such that $S'\vec{x} \multimap S(C\vec{x})$.

We observe that Q_j is contained in S': If Q_j accepts $\vec{t}$ then Q_i accepts $C\vec{t}$, and so does S; By cutting this against the above equivalence we obtain that S' accepts $\vec{t}$. So we have S' in S_j, and we select this conjunct on the left hand-side. We now have to derive $S'\vec{x} \vdash S(C\vec{x})$ which is simply the other direction of the above equivalence.

□

As for the corresponding proof about finite automata, this proof yields a (naive) decision procedure: there is only a finite number of invariants to try, and it is decidable to check the final premises, as well as the transition premises since their form is very limited. As for multi-simulation on automata, the full invariant considered in our completeness theorem is often unnecessarily large, but more economic techniques apply equally well on Q-states.

Unfortunately, it is not always possible to obtain Q-states. We propose a partial procedure for computing them, then discuss in which cases it might fail.

Algorithm 5.1 (Partial procedure for computing Q-states)**.** *Let P and Q be regular predicates, and (P_i) be the states of P. We denote by P_i^* a reordering of the arguments of P_i,* i.e., *P_i^* is $(\lambda(x_k)_k.\ P_i(x_{\sigma(k)})_k)$ for some permutation σ. Note that the characterization of states of Proposition 5.23, can be adapted to be of the form $\forall\vec{x}.\ (P_i\vec{x} \multimap (\vec{x} = C) \oplus (\exists\vec{y}.\ \vec{x} = C'\vec{y} \otimes \exists\vec{y}'.\ P_j^*\vec{y}\vec{y}') \oplus \dots$ where C' is non-erasing. We shall use that form in the proof below.*

The algorithm generates Q-states of the form $(\lambda\vec{x}.\ \vec{x} = C)$ or $(\lambda\vec{x}.\ \exists\vec{y}.\ \vec{x} = C\vec{y} \otimes \exists\vec{z}.\ P_j^*\vec{y}\vec{z})$ for some state P_j and a non-erasing pattern C. Strictly speaking, we need to generalize slightly over that format in order to handle erasing transitions of Q: we allow extra vacuous abstractions at any position, but limit the total arity to not exceed that of the transitions of C. This is shallow, and can be ignored in the following by considering the non-erasing restriction of a transition of Q, and adjusting the corresponding decomposition afterwards.

We build a set of Q-states as the fixed point[6] *of the following transformation, starting with the singleton $\lambda\vec{x}.\ \exists\vec{y}.\ \vec{x} = \vec{y} \otimes P_0\vec{y}$. The transformation consists in computing a decomposition of the right form for each P'_i of our tentative set of Q-states and each transition C_Q of Q, and adding the components of the decomposition to our collection:*

- *If P'_i is of the form $(\lambda\vec{x}.\ \vec{x} = C)$ then $P'_i(C_Q\vec{x})$ is provably equivalent to some $\vec{x} = C'$ if C_Q and C are compatible, which degenerates into $\mathbf{1}$ when $C_Q = C$ and $\vec{x}$ is empty; and it is equivalent to $\mathbf{0}$ if the patterns are incompatible. In both cases we have a valid decomposition, empty in the second case.*

- *Otherwise, our Q-state P' is of the form $(\lambda\vec{x}.\ \exists\vec{y}.\ \vec{x} = C\vec{y} \otimes \exists\vec{z}.\ P_i^*\vec{y}\vec{z})$.*

  - *If C_Q and C are incompatible, $P'(C_Q\vec{x})$ is simply equivalent to $\mathbf{0}$.*
  - *If C_Q has no rigid structure, it is enough to observe that $P'(C_Q\vec{x}) \multimap P'^*(\vec{x})$.*

[6] The iteration might diverge, if there is no finite fixed point.

– *If C has no rigid structure, $P'(C_Q\vec{x})$ is equivalent to $\exists\vec{z}.\ P_i^*(C_Q\vec{x})\vec{z}$. The predicate P_i is equivalent to its characterization as a state,* i.e., *the sum of all its transitions and final patterns: $P_i\vec{x'} \multimap (\oplus_j\ F_j\vec{x'}) \otimes (\oplus_k\ T_k\vec{x'})$. We decompose recursively[7] each $F_j^*(C_Q\vec{x})\vec{z}$ and $T_k^*(C_Q\vec{x})\vec{z}$ as a sum of Q-states, and manipulate the results to obtain a decomposition for $P'(C_Q\vec{x})$.*

Our $P'(C_Q\vec{x})$ is equivalent to $\exists\vec{z}.\ \oplus_k\ (P''_k\vec{xz})$, that is $\oplus_k\ \exists\vec{z}.\ (P''_k\vec{xz})$. It remains to adapt the disjuncts into well-formed Q-states. We only show how to treat the case of a transition clause P''_k, the treatment of a final clause being a particular case. We start with:

$$\exists\vec{z}.\ \exists\vec{y'}.\ (\vec{x},\vec{z}) = C'\vec{y'} \otimes \exists\vec{z'}.\ P_j^*\vec{y'}\vec{z'}$$

Splitting C' into (C'_1, C'_2) and $\vec{y'}$ into $(\vec{y'}_1, \vec{y'}_2)$ accordingly, we obtain:

$$\exists\vec{y'}_1.\ \vec{x} = C'_1\vec{y'}_1 \otimes \exists\vec{z}\exists\vec{y'}_2\exists\vec{z'}.\ \vec{z} = C'_2\vec{y'}_2 \otimes P_j^*\vec{y'}_1\vec{y'}_2\vec{z'}$$

Finally, we can remove the useless information about $\vec{z}$, without loosing the equivalence:

$$\exists\vec{y'}_1.\ \vec{x} = C'_1\vec{y'}_1 \otimes \exists\vec{y'}_2\exists\vec{z'}.\ P_j^*\vec{y'}_1\vec{y'}_2\vec{z'}$$

– *When C_Q and C both have some rigid structure, then $C_Q\vec{x} = C\vec{y}$ can be decomposed into $\vec{x}_1 = C'\vec{y}_1 \otimes \vec{y}_2 = C'_Q\vec{x}_2$, where $\vec{x}_1, \vec{x}_2$ (resp. $\vec{y}_1, \vec{y}_2$) is a partition of $\vec{x}$ (resp. $\vec{y}$). This decomposition is obtained by destructing the common rigid structure of C and C_Q, aggregating in C' (resp. C'_Q) the residual constraints corresponding to branches where C_Q (resp. C) becomes flexible first.*

So we have an equivalence between $P'(C_Q\vec{x})$ and:

$$\exists\vec{y}_1\vec{y}_2.\ \vec{x}_1 = C'\vec{y}_1 \otimes \vec{y}_2 = C'_Q\vec{x}_2 \otimes \exists\vec{z}.\ P_i^*\vec{y}_1\vec{y}_2\vec{z}$$

Or simply:

$$\exists\vec{y}_1.\ \vec{x}_1 = C'\vec{y}_1 \otimes \exists\vec{z}.\ P_i^*\vec{y}_1(C'_Q\vec{x}_2)\vec{z}$$

We recursively[8] compute the decomposition of P_i^* for the pattern $(\mathcal{I}_{|\vec{y}_1|}, C'_Q, \mathcal{I}_{|\vec{z}|})$. As before, we shall obtain a decomposition of P' from that of P_i^*. We detail the case of transition clauses, for which we obtain a disjunct of the following form:

$$\exists\vec{y}_1.\ \vec{x}_1 = C'\vec{y}_1 \otimes \exists\vec{z}.\ \exists\vec{y'_1}\exists\vec{y'_2}\exists\vec{y'_z}.\ (\vec{y}_1,\vec{x}_2,\vec{z}) = (C_1\vec{y'_1}, C_2\vec{y'_2}, C_z\vec{y'_z}) \otimes \exists\vec{z'}.\ P_j^*\vec{y'_1}\vec{y'_2}\vec{y'_z}\vec{z'}$$

We combine patterns:

$$\exists\vec{y'_1}\vec{y'_2}.\ (\vec{x}_1,\vec{x}_2) = (C'(C_1\vec{y'_1}), C_2\vec{y'_2}) \otimes \exists\vec{z}.\ \exists\vec{y'_z}.\ \vec{z} = C'_z\vec{y'_z} \otimes \exists\vec{z'}.\ P_j^*\vec{y'_1}\vec{y'_2}\vec{y'_z}\vec{z'}$$

[7] This recursive decomposition can loop if there is a cycle of trivial transitions in the states of P.

[8] This can create a loop if C'_Q does not decrease, *i.e.,* if C only has rigid structure on components where C_Q does not.

And finally remove the constraint on hidden variables $\vec{z}$, to obtain a decomposition of the right form:

$$\exists \vec{y_1} \vec{y_2}.\ (\vec{x_1}, \vec{x_2}) = (C'(C_1 \vec{y_1}), C_2 \vec{y_2}) \otimes \exists \vec{y_z}.\ \exists \vec{z'}.\ P^*_j \vec{y_1} \vec{y_2} \vec{y_z} \vec{z'}$$

As is visible in the definition, several problems can cause the divergence of our algorithm. We propose some constraints under which it terminates, but also show why it is interesting in a more general setting.

Proposition 5.27. *Let P be a regular predicate such that its states have an arity of at most one, and there is no cycle of ϵ-transitions in it. Then it admits Q-states for any regular Q. Hence the derivability of $\forall x.\ Qx \multimap Px$ is decidable, and holds whenever $Q \subseteq P$.*

Proof. We only have to check that P admits Q-states, the rest being a consequence of Theorem 5.26. Algorithm 5.1 always return valid Q-states by its definition, we show that it terminates under the assumptions on P. The algorithm proceeds by iterating a transformation until a fixed point is reached. We check that the transformation is well-defined, and that the fixed point is reached in a finite number of iterations.

There are two possible loops in the computation of a decomposition, in the last two cases. In the case where C is trivial, there is a recursive call on the same C_Q, but on a different state of P: this can loop only if there is a cycle of trivial transitions in P, which is excluded by hypothesis in our case. In the case where C and C_Q are non-trivial, there is a recursive call on C'_Q, which may not be smaller than C_Q in general: in our case, since C is unary, some structure is necessarily removed from C_Q.

It remains to check that the fixed point is reached in a finite number of iterations. It follows from the observation that the algorithm can only generate a finite number of distinct formulas of the right form, because generated patterns have their height bounded by the maximum height of the patterns of P. This is true with the initial Q-state which uses a trivial pattern. It is preserved by the decomposition algorithm, the only non-trivial case being the last one where we compose $C'C_1$. But since P is unary and C is non-erasing, C has at most one input variable $\vec{y} = y$. Hence, either C_Q forces some structure on y, in which case C' has no input ($\vec{y_2} = y$, $\vec{y_1}$ and C_1 are empty), or C_Q does not constrain y, in which case $\vec{y_2}$, $\vec{x_2}$ and C'_Q are empty, and $P^*_i y$ simply decomposes to itself (C_1 is an identity). In both cases $C'C_1$ has the height of C' which is less than that of C, itself less than the maximum height of the patterns of P. □

A regular formula constrained as in the previous proposition is not much more than a finite automata: we have essentially shown that Theorem 5.17 is a particular case of the results on regular formulas. A noticeable difference is the ϵ-acyclicity condition: there is no difficulty in extending the work on finite automata to handle ϵ-transitions. In fact, we conjecture that this assumption can also be removed from the previous result. But it would require a much more complex algorithm for computing the Q-states: the idea is that in order compute the decomposition of a state which belongs to an ϵ-cycle, one has to decompose at the same time all states of the cycle and merge their decompositions, using an induction rule to handle the cycle in the associated derivation of equivalence.

Example 5.28. In Proposition 5.27, the condition on the arity of P is essential. We show a simple binary example where our procedure diverges. Consider the regular predicates P and Q resulting from the following definition:

$$p\ (s^2 x)\ (s\ y) \triangleq p\ x\ y \qquad q\ (sx)\ (sy) \triangleq \ldots$$

We compute the Q-states of P using Algorithm 5.1. We start with p, and there is only one transition to consider: $(\lambda xy.sx, \lambda xy.sy)$. This is a rigid-rigid case, the decomposition of $p\ (sx)\ (sy)$ yields $\exists x' \exists y'.\ x = sx' \otimes y = y' \otimes p\ x'\ y'$. This new Q-state has to be decomposed for the same transition, and we obtain $\exists x' \exists y'.\ x = s^2 x' \otimes y = y' \otimes p\ x'\ y'$. The same pattern keeps applying, and new Q-states keep being generated.

Example 5.29. Consider the definitions of *nat* and $half$ from Example 5.10, translated as fixed points. They are regular predicates. Let $Q = \lambda x.\ \exists h.\ half\ x\ h$, $Q' = \lambda h.\ \exists x.\ half\ x\ h$. The algorithm successfully computes Q-states and Q'-states for *nat*. The obtained Q-states are: $\lambda x.\ nat\ x$, $\lambda x \lambda h.\ nat\ x$ and $\lambda x \lambda h.\ \mathbf{1}$. The Q'-states are: $\lambda h.\ nat\ h$, $\lambda x \lambda h.\ nat\ h$ and $\lambda x \lambda h.\ \mathbf{1}$. This yields μMALL proofs for $\forall x.\ (\exists h.\ half\ x\ h) \multimap nat\ x$ and $\forall h.\ (\exists x.\ half\ x\ h) \multimap nat\ h$.

Example 5.30. In Example 5.10, we gave an informal proof of the totality of $half$ by seeing *nat* and $half$ as finite automata. We can now avoid that step and obtain directly a derivation. The states of $H = \lambda x. \exists h.\ half\ x\ h$ are:

$$\begin{aligned} H_0 &\equiv \lambda x. \exists h.\ half\ x\ h \\ H_1 &\equiv \lambda x \lambda h.\ half\ x\ h \end{aligned}$$

Its *nat*-states can be obtained by our procedure:

$$\begin{aligned} H'_0 &\equiv \lambda x.\ \exists h.\ half\ x\ h \\ H'_1 &\equiv \lambda x.\ x = 0 \\ H'_2 &\equiv \lambda x.\ \exists p.\ x = sp \otimes \exists h.\ half\ p\ h \\ H'_3 &\equiv \lambda x.\ \mathbf{1} \end{aligned}$$

Starting from H'_0, and taking the successor transition of *nat*, we obtain H'_1 and H'_2 corresponding to the two transitions of H_1 that are compatible with the successor. Finally, H'_3 is obtained for the decomposition of all others against the zero transition. Notice that it is crucial that our algorithm eliminates the information learned about h as some constraints are applied on x, otherwise we could not obtain a finite number of *nat*-states.

Applying the proof of completeness, we essentially obtain the following invariant:

$$S := \lambda x.\ (\exists h.\ half\ x\ h)\ \&\ (x = 0 \oplus \exists y.\ x = s\ y \otimes \exists h.\ half\ y\ h)$$

Completing the proof is rather simple once we have that invariant. Let us write S as $H\ \&\ BH$ where B is the body of the fixed point *nat*: it trivially implies the expected theorem, that is H; for the base case of invariance we must show $H0\ \&\ (0 = 0 \oplus \ldots)$;

the heredity case is more interesting:

$$\dfrac{\dfrac{\vdash H(s0) \quad Hy \vdash H(s^2y)}{BHx \vdash H(sx)}}{\dfrac{Hx \;\&\; BHx \vdash H(sx)}{Hx \;\&\; BHx \vdash H(sx) \;\&\; BH(sx)}} \quad \dfrac{\overline{Hx \vdash Hx}}{Hx \;\&\; BHx \vdash BH(sx)}$$

Notice here the useful invariant scheme $\&_{i=0..n}\ B^i S$, which yields an induction with access to the n predecessors.

5.2.2 Beyond cyclic proofs

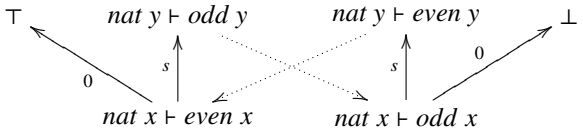
As said in Section 2.3.2, cyclic proofs are appealing from an automated reasoning perspective, as they avoid the invention of invariants, but they are very weak. It is our hope that the work presented in this chapter eventually leads to an useful theorem proving technique that would keep the practical aspect of cyclic proofs but be much more powerful, in particular be complete for inclusions of automata, and still meaningful for regular formulas.

On the particular inclusion problem, cyclic proofs seem related to simulations, associating one state with another. The proofs obtained from our completeness theorems use invariants that express the less restrictive notion of multi-simulation, where one state can be related to several. This offers an interesting trade-off between expressiveness and the subformula property, since the invariants under consideration are made of subformulas of the goal.

It is interesting to attempt to present our proofs in an extended cyclic style, which takes into account failures and alternatives, as well as loops between alternatives. For example:

$$\dfrac{\boxed{\dfrac{\dfrac{}{\vdash even\ 0} \quad \dfrac{\dfrac{\infty}{nat\ y \vdash odd\ y}}{nat\ y \vdash even\ (sy)}}{nat\ x \vdash even\ x}} \oplus \boxed{\dfrac{\dfrac{\bot}{\vdash odd\ 0} \quad \dfrac{\dfrac{\infty}{nat\ y \vdash even\ y}}{nat\ y \vdash odd\ (sy)}}{nat\ x \vdash odd\ x}}}{nat\ x \vdash even\ x \oplus odd\ x}$$

Establishing the correctness of such objects is complex, but we probably have most of the tools at hand. It is likely to be related to the inclusion of *nat* in the underlying automata:



Our work on regular formulas shows that there are two main steps when proving an implication of regular formulas $\forall \vec{x}.\ P\vec{x} \multimap Q\vec{x}$: first, one should obtain the P-states of Q; then one should try to build invariants from those P-states. The first step might fail, the second one is decidable. It seems possible to interpret this in terms of proof-search. The computation of the P-states would correspond to an exhaustive proof-search for $P\vec{x} \vdash Q\vec{x}$, only unfolding fixed points and detecting loops. This is visible on

the example of the totality of $half$:

$$\dfrac{\dfrac{}{\vdash half\ 0\ H}\qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 = 0}\quad \dfrac{\bot}{\vdash sz = 0}}{nat\ y \vdash y = 0}}{nat\ y \vdash sy = s0 \otimes H = 0} \oplus \dfrac{\dfrac{\bot}{\vdash 0 = sZ \otimes \ldots}\quad \dfrac{\dfrac{\infty}{nat\ z \vdash half\ z\ H'}}{nat\ z \vdash sz = sZ \otimes half\ Z\ H'}}{nat\ y \vdash y = sZ \otimes half\ Z\ H'}}{nat\ y \vdash sy = s^2Z \otimes H = sH' \otimes half\ Z\ H'}}{nat\ y \vdash half\ (sy)\ H}}{nat\ x \vdash half\ x\ H}$$
$$\dfrac{}{nat\ x \vdash \exists h.\ half\ x\ h}$$

If that exploration succeeds, the information about loops, failed and proved branches would be checked for correctness. This second step, when successful, could build a proof by explicit induction. It would also be interesting to study how to generate a counter-example in case of failure.

5.3 Conclusion

We have shown that μMALL offers enough expressiveness for reasoning about automata inclusions. More importantly, the connection was natural, and brought new observations in both automata and proof theory. We have seen that the logical treatment of Büchi automata raises several interesting problems; we consider it as an important direction for further exploration of μMALL. The study of tree automata would also be a natural next step, calling for a similar extension of regular formulas. Finally, another direction for future work is to consider the implementation of proof-search ideas based on regular formulas.

An advantage of having adapted automata techniques to proof theory is that they should extend well to other features of logic. For example, it would be interesting to consider higher-order terms and unification. Together with support for generic quantification (which is the topic of the next chapter) this would extend the scope of our study, for example, to process calculi such as π-calculus, whose labeled transition system involves bindings.

Chapter 6

Reasoning about generic judgments

We have seen that logics supporting fixed points are good frameworks for representing objects and reasoning about the relationships between these objects. We have lead a proof-theoretical study of such a logic, namely μMALL, and discovered some important structure of its derivations. Proof-theory offers a particularly interesting methodology to develop and study such frameworks, as it is usually modular. For example, the structure of focused derivations for μMALL has been extended to μLJ. We shall see another example of this modularity in this chapter, by showing how to integrate the notion of *generic quantification* into sequent calculus in a completely orthogonal way.

Generic quantification has been introduced by Miller and Tiu [MT03a, MT05] in order to support reasoning about specifications involving variable bindings. Such specifications are very common in computer science: type systems, programming languages, logics, etc. But their formal treatment is delicate, and is still subject to active research. Miller and Tiu's simple design allows for equally simple proofs expressing rich reasoning about generic variables. Unfortunately, that initial formulation of generic quantification is not so orthogonal to other features of logic: it turns out to have a poor interaction with fixed points, which limits the expressiveness of the resulting logic. In this chapter, we come back to that early design and refine it. We shall work with μLJ as it is our practical target, but it would work equally well, for example, in μMALL.

In our introductory Chapter 2, we have outlined three increasingly demanding steps in that program: computing, model-checking and reasoning — from the proof-as-program viewpoint, representing objects, then finite and finally infinite behavior functions on those objects. In Section 6.1, we introduce the original design of generic quantification, showing how it addresses the finite behaviors but is limited concerning infinite ones. In Section 6.2 we propose the logic μLJ^∇ as a replacement that brings the expected expressiveness, by allowing a better interaction between fixed point constructions and generic quantification; we explore the meta-theory of μLJ^∇, notably giving evidence of its expressiveness. Finally, we illustrate how the logic can be practically used on a few significant examples in Section 6.3. Before concluding, we shall discuss related systems in Section 6.4.

6.1 The original design of ∇

We motivate the introduction of a new quantifier for expressing generic quantification, before formally presenting its proof-theoretic design.

6.1.1 Motivation

The *higher-order abstract syntax* (HOAS) approach [HL78, MN87, PE88] allows for elegant, high-level representations of objects involving variable binding, such as programs or formulas, but also of transformations and relations involving such objects, *e.g.,* evaluation or type checking. It basically consists in leveraging the notion of binding already present in logic to represent binding at the object-level. In the proof-as-program approach, one uses the abstraction of proofs terms. In the λ-tree approach, term-level abstractions are used, and generic variables are introduced by first-order universal quantifications. Tools such as Twelf [PS99] and λProlog [MNPS91] can be used to compute on HOAS specifications.

As we have seen in Chapter 2 with the example of negation as failure, when the adequacy of a representation is justified by an extra-logical argument, it might not be possible to reason properly about the represented objects from within the logic. The design of Twelf reflects this limitation by having two levels. The lower level is the logical framework (LF) used to represent objects. Twelf's metatheorem prover uses a different formalism which manipulates LF objects to establish properties about them, *e.g.,* termination or totality. Taking a different approach, Miller and Tiu developed the generic quantifier ∇ [MT03a, MT05] in order to obtain a proper logic which could both serve to represent and reason about HOAS specifications.

In order to see why Miller and Tiu needed to introduce a new quantifier, we shall consider the example of simple typing for Church-style λ-calculus:

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma \vdash M : T' \to T \quad \Gamma \vdash N : T'}{\Gamma \vdash MN : T} \qquad \frac{\Gamma, x : T \vdash Mx : T'}{\Gamma \vdash (\lambda x : T.\, Mx) : (T \to T')}$$

Following the higher-order abstract syntax approach, let us encode this specification as a least fixed point — or a definition, the distinction does not matter at this point. We assume two term types tm and ty, constants $app : tm \to tm \to tm$ and $abs : (tm \to tm) \to tm$ for representing terms, and $arrow : ty \to ty \to ty$ for types. We shall represent the context Γ as a list of pairs, using usual associated types and constructors. We define the predicate $of : (tm * ty)\ list \to tm \to ty \to o$, and write $\{\Gamma \vdash_{\Lambda^\to} M : T\}$ for $(of\ \Gamma\ M\ T)$ for better readability:

$$\begin{aligned}
\{\Gamma \vdash_{\Lambda^\to} X : T\} &\triangleq \langle X, T\rangle \in \Gamma \\
\{\Gamma \vdash_{\Lambda^\to} app\ M\ N : T\} &\triangleq \{\Gamma \vdash_{\Lambda^\to} M : arrow\ T'\ T\} \wedge \{\Gamma \vdash_{\Lambda^\to} N : T'\} \\
\{\Gamma \vdash_{\Lambda^\to} abs\ T\ M : arrow\ T\ T'\} &\triangleq \forall_{tm} x.\, \{\langle x, T\rangle :: \Gamma \vdash_{\Lambda^\to} Mx : T'\}
\end{aligned}$$

This natural encoding of typing judgments relies on the extra-logical observation that derivations of $\forall x.Px$ are derivations of Px for a generic x. This does not correspond, however, to the logical meaning of the universal quantifier, which is "for

all". To see why this is a problem, let us reason about our typing judgments. Consider, for example, the following observation about λ-calculus: if $(\lambda_\alpha x \lambda_\beta y.\ x)$ has type $\alpha \to \beta \to \beta$ then it must be that α and β are the same. That observation follows from a simple examination of the possible derivations of the typing judgment: it must start with two abstraction rules, introducing the variables x and y, then finish with a variable rule on x, which requires $\alpha = \beta$. In μLJ, that observation is stated as follows:

$$\forall\alpha\forall\beta.\ \{nil \vdash_{\Lambda\to} (abs\ \alpha\ (\lambda_{tm} x.\ abs\ \beta\ (\lambda_{tm} y.\ x))) : arrow\ \alpha\ (arrow\ \beta\ \beta)\} \supset \alpha = \beta$$

In order to prove it, we proceed by case analysis on the typing judgment, in which only the abstraction case is not absurd. We are left with an universal quantification on the left: we have to choose the variable that shall represent x. *If* we have two different constants of type *tm*, we can choose to represent x and y by them, which allows to conclude by case-analysis on the typing judgment that α and β must be equal. This does not correspond at all to the intuitive proof, because the left $\forall$ rule does not correspond to the meta-level "generic" reading of the right $\forall$ rule but to the logical "for all" meaning of the quantifier. In particular, it is possible to instantiate two universal quantifiers with the same term, *e.g.,* when deriving $\forall x \forall y.\ Pxy \vdash \forall x.\ Pxx$.

Miller and Tiu designed the ∇ quantifier to express logically the notion of generic quantification. As long as it is in positive position, ∇ behaves as $\forall$, hence it does not change the represented objects. But ∇ is self-dual: both its left and right introduction rules correspond to the introduction of a generic variable. We shall come back to our example after a formal presentation of the original design of generic quantification.

6.1.2 The logic $\mu\mathbf{LJ}^{\nabla_0}$

The ∇-quantifier was first introduced with the logic $FO\lambda^{\Delta\nabla}$ [MT03a], an extension of LJ with higher-order terms, definitions and ∇. That system was then extended to inductive and coinductive definitions, resulting in the logic LINC [MT03b, Tiu04]. We consider the closely related logic $\mu\mathrm{LJ}^{\nabla_0}$ given in Figure 6.1, featuring least and greatest fixed points and the ∇ quantifier. The difference between LINC and $\mu\mathrm{LJ}^{\nabla_0}$, *i.e.,* the difference between fixed points and definitions, is not relevant to the original design of generic quantification. We shall see, however, that working with fixed point formulas is conceptually important in our refinement of the system.

Let us introduce a few definitions and notations. *Generic contexts* are lists of typed term variables, denoted by σ or ζ. If σ is a generic context $(x_1 : \gamma_1, \ldots, x_n : \gamma_n)$ we denote by $\sigma \to \gamma$ the type $\gamma_1 \to \ldots \to \gamma_n \to \gamma$, and call it γ *lifted over* σ. Analogously, we talk of lifted variables when they have a lifted type. We also use a generic context as a list of terms, writing $(F\sigma)$ for $((Fx_1)\ldots x_n)$. Similarly, $f(x_i)_i$ stands for $((fx_1)\ldots x_n)$. As usual, type annotations are omitted for conciseness most of the time, but can be recovered from the context. In order to make type inference easier, we use the convention of naming x' a lifted version of x. Finally, when we write a formula $P\vec{x}$, it indicates that the variables $\vec{x}$ can occur in it, and assumes that they don't occur free in P anymore; conversely, we use the convention that when we write $(\lambda a.t)$, the variable a does not occur free in t.

Propositional intuitionistic logic

$$\frac{}{\Sigma; \Gamma, \sigma \rhd \bot \vdash G} \qquad \frac{}{\Sigma; \Gamma \vdash \sigma \rhd \top}$$

$$\frac{\Sigma; \Gamma, \sigma \rhd P\sigma, \sigma \rhd Q\sigma \vdash G}{\Sigma; \Gamma, \sigma \rhd P\sigma \wedge Q\sigma \vdash G} \qquad \frac{\Sigma; \Gamma \vdash \sigma \rhd P\sigma \quad \Sigma; \Gamma \vdash \sigma \rhd Q\sigma}{\Sigma; \Gamma \vdash \sigma \rhd P\sigma \wedge Q\sigma}$$

$$\frac{\Sigma; \Gamma, \sigma \rhd P\sigma \vdash G \quad \Sigma; \Gamma, \sigma \rhd Q\sigma \vdash G}{\Sigma; \Gamma, \sigma \rhd P\sigma \vee Q\sigma \vdash G} \qquad \frac{\Sigma; \Gamma \vdash \sigma \rhd P_i\sigma}{\Sigma; \Gamma \vdash \sigma \rhd P_0\sigma \vee P_1\sigma}$$

$$\frac{\Sigma; \Gamma \vdash \sigma \rhd P\sigma \quad \Sigma; \Gamma, \sigma \rhd Q\sigma \vdash G}{\Sigma; \Gamma, \sigma \rhd P\sigma \supset Q\sigma \vdash G} \qquad \frac{\Sigma; \Gamma, \sigma \rhd P\sigma \vdash \sigma \rhd Q\sigma}{\Sigma; \Gamma \vdash \sigma \rhd P\sigma \supset Q\sigma}$$

First-order structure

$$\frac{\Sigma, x' : \sigma \to \gamma; \Gamma, \sigma \rhd P\sigma(x'\sigma) \vdash G}{\Sigma; \Gamma, \sigma \rhd \exists_\gamma x.P\sigma x \vdash G} \qquad \frac{\Sigma \vdash t' : \sigma \to \gamma \quad \Sigma; \Gamma \vdash \sigma \rhd P\sigma(t'\sigma)}{\Sigma; \Gamma \vdash \sigma \rhd \exists_\gamma x.P\sigma x}$$

$$\frac{\Sigma \vdash t' : \sigma \to \gamma \quad \Sigma; \Gamma, \sigma \rhd P\sigma(t'\sigma) \vdash G}{\Sigma; \Gamma, \sigma \rhd \forall_\gamma x.P\sigma x \vdash G} \qquad \frac{\Sigma, x' : \sigma \to \gamma; \Gamma \vdash \sigma \rhd P\sigma(x'\sigma)}{\Sigma; \Gamma \vdash \sigma \rhd \forall_\gamma x.P\sigma x}$$

$$\frac{\{(\Sigma; \Gamma \vdash G)\theta : (\lambda\sigma.u\sigma)\theta \doteq (\lambda\sigma.v\sigma)\theta\}}{\Sigma; \Gamma, \sigma \rhd u\sigma = v\sigma \vdash G} \qquad \frac{}{\Sigma; \Gamma \vdash \sigma \rhd t\sigma = t\sigma}$$

$$\frac{\Sigma; \Gamma, (\sigma, x) \rhd P\sigma x \vdash G}{\Sigma; \Gamma, \sigma \rhd \nabla x.P\sigma x \vdash G} \qquad \frac{\Sigma; \Gamma \vdash (\sigma, x) \rhd P\sigma x}{\Sigma; \Gamma \vdash \sigma \rhd \nabla x.P\sigma x}$$

Fixed points

$$\frac{\Sigma; \Gamma, \sigma \rhd S\,(t\sigma) \vdash P \quad x; \cdot \rhd BS\,x \vdash \cdot \rhd S\,x}{\Sigma; \Gamma, \sigma \rhd \mu B(t\sigma) \vdash P} \qquad \frac{\Sigma; \Gamma \vdash \sigma \rhd B(\mu B)(t\sigma)}{\Sigma; \Gamma \vdash \sigma \rhd \mu B(t\sigma)}$$

$$\frac{\Sigma; \Gamma, \sigma \rhd B(\nu B)(t\sigma) \vdash P}{\Sigma; \Gamma, \sigma \rhd \nu B(t\sigma) \vdash P} \qquad \frac{\Sigma; \Gamma \vdash \sigma \rhd S\,(t\sigma) \quad x; \cdot \rhd S\,x \vdash \cdot \rhd BS\,x}{\Sigma; \Gamma \vdash \sigma \rhd \nu B(t\sigma)}$$

$$\frac{}{\Sigma; \Gamma, \sigma \rhd \mu B(t\sigma) \vdash \sigma \rhd \mu B(t\sigma)} \qquad \frac{}{\Sigma; \Gamma, \sigma \rhd \nu B(t\sigma) \vdash \sigma \rhd \nu B(t\sigma)}$$

Figure 6.1: Inference rules for $\mu\mathrm{LJ}^{\nabla_0}$

The formulas of $\mu\mathrm{LJ}^{\nabla_0}$ are obtained by adding the ∇ quantifier to the constructions of μLJ:

$$\begin{array}{rcl} P & ::= & P \wedge P \mid P \vee P \mid P \supset P \mid \top \mid \bot \\ & \mid & \exists_\gamma x.Px \mid \forall_\gamma x.Px \mid \nabla_\gamma x.Px \mid s \overset{\gamma}{=} t \mid s \overset{\gamma}{\neq} t \mid \mu_{\gamma_1 \ldots \gamma_n} B\vec{t} \mid \nu_{\gamma_1 \ldots \gamma_n} B\vec{t} \end{array}$$

The sequents of $\mu\mathrm{LJ}^{\nabla_0}$ are obtained by extending the structure of intuitionistic sequents with a local generic context surrounding each formula, written $\sigma \triangleright P\sigma$. That context binds *generic variables* in the formula. A formula and its context form a *generic judgment*, denoted by G.

The deduction rules of $\mu\mathrm{LJ}^{\nabla_0}$ are given in Figure 6.1. Propositional logical rules are extended orthogonally to generic contexts. Variables are introduced in generic contexts by the ∇ quantifier, in the same way on both sides. Unlike propositional connectives, the first-order quantifiers $\forall$ and $\exists$ interact with the generic context, causing a lifting of terms and term variables. For example, in the right $\forall$ rule, if x has type γ then x' has type $\sigma \to \gamma$. This treatment preserves the global aspect of universal variables, while expressing the local dependency on the generic context. Finally, the generic context is interpreted as a λ-abstraction in the equality rules.

At this point, we have described how the logic treats *finite behavior* formulas, *i.e.,* those that do not involve fixed points. In that setting, the deduction rules can be read as equivalences, for example $\nabla x.\ Px \wedge Qx \equiv (\nabla x.\ Px) \wedge (\nabla x.\ Qx)$, which allow to eliminate ∇ by pushing it down through logical connectives, eventually disappearing, for example thanks to $(\nabla x.\ ux = vx) \equiv u = v$. A similar observation is that any proof of a finite behavior can be η-expanded until it does not use the axiom rule anymore, but only $\top, \bot$ and equality rules as initial ones. Both observations imply that there is no need to ever compare two generic contexts: the question of the identity of formulas is irrelevant.

We can extend the setting where identity does not matter by considering proofs involving fixed points unfoldings, naturally extended to generic contexts, but no axiom:

$$\frac{\Gamma, \sigma \triangleright B(\mu B)(t\sigma) \vdash G}{\Gamma, \sigma \triangleright \mu B(t\sigma) \vdash G} \qquad \frac{\Gamma \vdash \sigma \triangleright B(\mu B)(t\sigma)}{\Gamma \vdash \sigma \triangleright \mu B(t\sigma)}$$

It is in such a framework that Miller and Tiu obtained a fully declarative specification of finite π-calculus and its bisimulation [Tiu05], which is probably the best demonstration of the success of $FO\lambda^{\Delta\nabla}$. It is also in that setting that the previous example lies.

Example 6.1. Let us come back to the example of simply typed λ-calculus, when generic quantification is used to represent the introduction of a variable in the abstraction rule:

$$\begin{array}{rcl} \{\Gamma \vdash_{\Lambda^\to} X : T\} & \triangleq & \langle X, T\rangle \in \Gamma \\ \{\Gamma \vdash_{\Lambda^\to} app\ M\ N : T\} & \triangleq & \{\Gamma \vdash_{\Lambda^\to} M : arrow\ T'\ T\} \wedge \{\Gamma \vdash_{\Lambda^\to} N : T'\} \\ \{\Gamma \vdash_{\Lambda^\to} abs\ T\ M : arrow\ T\ T'\} & \triangleq & \nabla x.\ \{\langle x, T\rangle :: \Gamma \vdash_{\Lambda^\to} Mx : T'\} \end{array}$$

We consider the same observation as before, and derive it in $\mu\mathrm{LJ}^{\nabla_0}$. The essential

steps of the derivation closely correspond to the informal idea of the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha,\beta;x,y \rhd \langle x,\beta\rangle = \langle y,\beta\rangle \vdash \alpha = \beta}\,{=}L \quad \dfrac{\dfrac{}{\alpha;\vdash \alpha = \alpha}}{\alpha,\beta;x,y \rhd \langle x,\beta\rangle = \langle x,\alpha\rangle \vdash \alpha = \beta}\,{=}L}{\alpha,\beta;x,y \rhd \langle x,\beta\rangle \in \langle y,\beta\rangle :: \langle x,\alpha\rangle :: nil \vdash \alpha = \beta}}{\alpha,\beta;x,y \rhd \{\langle y,\beta\rangle :: \langle x,\alpha\rangle :: nil \vdash_{\Lambda\to} x : \beta\} \vdash \alpha = \beta}}{\alpha,\beta;x \rhd \{\langle x,\alpha\rangle :: nil \vdash_{\Lambda\to} (abs\,\beta\,(\lambda_{tm}y.\ x)) : arrow\,\beta\,\beta\} \vdash \alpha = \beta}}{\alpha,\beta;\cdot \rhd \{nil \vdash_{\Lambda\to} (abs\,\alpha\,(\lambda_{tm}x.\ abs\,\beta\,(\lambda_{tm}y.\ x))) : arrow\,\alpha\,(arrow\,\beta\,\beta)\} \vdash \alpha = \beta}}{;\vdash \forall\alpha\forall\beta.\ \{nil \vdash_{\Lambda\to} (abs\,\alpha\,(\lambda_{tm}x.\ abs\,\beta\,(\lambda_{tm}y.\ x))) : arrow\,\alpha\,(arrow\,\beta\,\beta)\} \supset \alpha = \beta}$$

That derivation proceeds by case analysis, *i.e.,* left fixed point unfoldings. For each abstraction case, a new generic variable is introduced in the local context. Notice how there are binders all the way down: the object-level binder encoded as an abstraction in higher-order terms is manipulated in the specification by means of generic quantification, and changed during deduction into a sequent-level abstraction as part of the generic context. Finally, there are two possibilities when looking up the typing context: $\langle x,\beta\rangle$ is either $\langle y,\beta\rangle$ or $\langle x,\alpha\rangle$. The former possibility is ruled out, as it requires to identify two generic variables, *i.e.,* to unify distinct bound variables. The latter possibility implies $\alpha = \beta$.

The logical treatment of infinite behaviors, through the rules of (co)induction and axioms, is more problematic. The axioms on fixed points require an exact match (modulo α-equivalence) of the generic contexts. The extended unfolding rules state that liftings of a fixed point unfold just like the original version. The induction rule seems less natural, essentially stating that liftings of the invariants of a fixed point are invariants of its liftings. Tiu noticed [Tiu04] that it is too weak, since it does not allow any modification of the generic context when inducting under it. As a consequence, it is impossible to prove things like $\forall x.\ (\nabla a.\ nat\ x) \supset nat\ x$. This seems unfortunate because *nat*, defined as a least fixed point, clearly does not rely on the generic context.

There are two possible ways to address this lack of expressiveness. The first one is to change the axiom rule. This is essentially what is done in [Tiu06] where the author adds structural rules for generic contexts; we shall come back to this approach in Section 6.4. The second possibility is to work with the (co)induction rule, which is the natural thing to consider in this thesis. In the following, we are going to fix the logic in order to be able to express the missing properties, such as $\forall x.\ (\nabla a.\ nat\ x) \supset nat\ x$, by relying on the structure of formulas. It is thus very important that we work in a logic where every construction is fully defined, *i.e.,* canonical. This excludes atoms, *i.e.,* predicate constants, as well as retracts (*i.e.,* arbitrary fixed points). If one wants to obtain $\forall x.\ (\nabla a.\ px) \supset px$ for a predicate constant p, there is no other option than adding it as a rule in the logic, whereas with a canonical fixed point the theorem could be obtained by (co)induction. Not considering atoms is motivated practically, because users of a logic usually work on defined notions, but also theoretically, as we find it important for the logician to study what can be derived from the simplest definition before playing the game of choosing which axioms to force uniformly. As we shall see, this approach allows to preserve the essence of what we call *minimal* generic quantification for the absence of any structural rule for generic contexts.

6.2 μLJ$^{\nabla}$: treating ∇ as a non-logical connective

The self-duality of generic quantification, as well as its ability to commute with almost all connectives of μLJ$^{\nabla_0}$, suggests that it might not be a *logical connective* but rather a *defined* one, like negation in classical logics. The logic μLJ$^{\nabla}$ is the result of this observation. We shall define a formula transformation that produces formulas where the ∇ quantifiers only occur above bound predicate variables inside fixed point constructions. This is in slight contrast with the classical negation which can be eliminated statically even inside fixed points, because of the monotonicity constraint. The important point remains: since ∇ does not occur anymore at toplevel in a sequent, it loses its logical role.

The logic μLJ$^{\nabla}$ has the same formulas as μLJ$^{\nabla_0}$ but the same rules as μLJ (see Figure 2.1). The sequents of μLJ$^{\nabla}$ are standard intuitionistic sequents. The rules for fixed points are unchanged, but the implicit elimination of toplevel ∇ quantifiers which might occur in BS will play a critical role in (co)induction:

$$\frac{\Gamma, S\,t \vdash P \quad BS\,x \vdash S\,x}{\Gamma, \mu B t \vdash P} \qquad \frac{\Gamma \vdash B(\mu B)t}{\Gamma \vdash \mu B t}$$

We define the connective ∇ by identifying a formula F at toplevel in a sequent with $\phi(F)$ where ∇ does not occur anymore except inside fixed point bodies. The inductive definition of ϕ is given on Figure 6.2. The transformation is parametrized by two contexts initially empty, and written $\phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma)$. Here, σ is a generic context, *i.e.,* a list of term variables. The context Γ contains associations of the form $\langle p, \sigma, p'\rangle$ where p is a predicate variable of type $\gamma \to o$, $\sigma = x_1 : \gamma_1, \ldots, x_n : \gamma_n$ is a generic context and p' is an other predicate variable of type $(\gamma_1 \to \ldots \to \gamma_n \to \gamma) \to o$. The support of Γ, written $|\Gamma|$, is the variables p. As indicated by the notation in $P|\Gamma|\sigma$, the predicate variables of $|\Gamma|$ and the term variables of σ may occur in the original formula. These occurrences are bound by the two contexts. In the transformed formula only the p' will be found. Finally, the order does not matter in Γ, unlike in σ.

We shall write ϕ applied not only to formulas but also to predicates and predicate operators, by defining:

$$\phi^{\Gamma}_{\sigma}(\lambda x.\, Px) \equiv \lambda x'.\, \phi^{\Gamma}_{\sigma}(P(x'\sigma)) \qquad \phi^{\Gamma}_{\sigma}(\lambda p.\, Bp) \equiv \lambda p'.\, \phi^{\Gamma,\langle p,\sigma,p'\rangle}_{\sigma}(Bp)$$

In fact, the definition of ϕ can be reduced to these two definitions extended in an orthogonal way to all logical connectives. For doing so, one should carefully separate the binding and logical aspects of $\forall, \exists, \mu$ and ν.

Example 6.2. The elimination of ∇ on finite-behavior formulas described for μLJ$^{\nabla_0}$ in Section 6.1.2 is now an identity. In μLJ$^{\nabla}$ the two following formulas are identified:

$$\nabla x.\forall y.\nabla z.y = z \supset \bot \quad \equiv \quad \forall y'.(\lambda xz.y'x) = (\lambda xz.z) \supset \bot$$

Example 6.3. We define the well-formedness of terms in a purely abstractive language, assuming constants $nil : lst$, $cons : tm \to lst \to lst$ and $abs : (tm \to tm) \to tm$, as follows:

$$mem \stackrel{def}{=} \mu(\lambda M \lambda x \lambda \Gamma.\, \exists hd \exists tl.\, \Gamma = (cons\ hd\ tl) \wedge (x = hd \vee M\ x\ tl))$$

$$term \stackrel{def}{=} \mu(\lambda T \lambda \Gamma \lambda t.\, mem\ t\ \Gamma \vee \exists f.\, t = abs\ f \wedge \nabla_{tm} x.\, T\ (cons\ x\ \Gamma)\ (f\ x))$$

$$\phi^{\Gamma}_{\sigma}(\nabla x.\ P|\Gamma|\sigma x) \equiv \phi^{\Gamma}_{\sigma,x}(P|\Gamma|\sigma x)$$

$$\phi^{\Gamma}_{\sigma}(\forall x.\ P|\Gamma|\sigma x) \equiv \forall x'.\ \phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma(x'\sigma)) \quad \phi^{\Gamma}_{\sigma}(\exists x.\ P|\Gamma|\sigma x) \equiv \exists x'.\ \phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma(x'\sigma))$$

$$\phi^{\Gamma}_{\sigma}(\mu(\lambda p \lambda \vec{x}.\ B|\Gamma|p\sigma\vec{x})(\vec{t\sigma})) \equiv \mu(\lambda p' \lambda \vec{x'}.\ \phi^{\Gamma,\langle p,\sigma,p'\rangle}_{\sigma}(B|\Gamma|p\sigma(\vec{x'\sigma})))\vec{t}$$

$$\phi^{\Gamma}_{\sigma}(\nu(\lambda p \lambda \vec{x}.\ B|\Gamma|p\sigma\vec{x})(\vec{t\sigma})) \equiv \nu(\lambda p' \lambda \vec{x'}.\ \phi^{\Gamma,\langle p,\sigma,p'\rangle}_{\sigma}(B|\Gamma|p\sigma(\vec{x'\sigma})))\vec{t}$$

$$\phi^{\Gamma,\langle p,\sigma,p'\rangle}_{\sigma\sigma'}(p(t\sigma\sigma')) \equiv \nabla\sigma'.\ p'(\lambda\sigma.\ t\sigma\sigma')$$

$$\phi^{\Gamma}_{\sigma}(u\sigma = v\sigma) \equiv u = v \quad \phi^{\Gamma}_{\sigma}(\top) \equiv \top \quad \phi^{\Gamma}_{\sigma}(\top) \equiv \top$$

$$\phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma \wedge Q|\Gamma|\sigma) \equiv \phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma) \wedge \phi^{\Gamma}_{\sigma}(Q|\Gamma|\sigma)$$

$$\phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma \vee Q|\Gamma|\sigma) \equiv \phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma) \vee \phi^{\Gamma}_{\sigma}(Q|\Gamma|\sigma)$$

$$\phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma \supset Q|\Gamma|\sigma) \equiv \phi^{\Gamma}_{\sigma}(P|\Gamma|\sigma) \supset \phi^{\Gamma}_{\sigma}(Q|\Gamma|\sigma)$$

Figure 6.2: The transformation ϕ defining ∇

It does not look very different from $\mu\mathbf{LJ}^{\nabla_0}$. But the ∇ inside the body of the fixed point *term* should be read as a suspended lifting, waiting for the instantiation of T. In a sense, the fixed point does not only define a predicate, but rather a family of liftings. Let us now consider the lifting of the definitions of *mem* and *term*. For readability we lift over an other type than *tm*, called γ:

$$\begin{array}{l}
\phi_{(y:\gamma)}(mem) \stackrel{def}{=} \\
\quad \mu(\lambda M' \lambda x' \lambda \Gamma'.\ \exists hd' \exists tl'.\ \Gamma' = (\lambda_\gamma y.\ cons\ (hd'\ y)\ (tl'\ y)) \wedge (x' = hd' \vee M'\ x'\ tl')) \\
\phi_{(y:\gamma)}(term) \stackrel{def}{=} \mu(\lambda T' \lambda \Gamma' \lambda t'.\ \phi_{(y:\gamma)}(mem)\ t'\ \Gamma' \vee \\
\quad \exists f'.\ t' = (\lambda_\gamma y.abs\ (f'y)) \wedge \nabla_{tm} x.\ T'\ (\lambda_\gamma y.\ cons\ x\ (\Gamma' y))\ (\lambda_\gamma y.\ f'yx))
\end{array}$$

In other words, the induction on $\phi_{(y:\gamma)}(term)$ corresponds to the following principle, where S is the candidate invariant:

$$\begin{array}{ll}
 & (\forall \Gamma' \forall x'.\ \phi_{(y:\gamma)}(mem)\ x'\ \Gamma' \supset S\ \Gamma'\ x') \\
\supset & (\forall \Gamma' \forall f'.\ \phi_{(x:tm)}(S)\ (\lambda x \lambda y.\ cons\ x\ (\Gamma' y))\ (\lambda x \lambda y.\ f'yx) \supset S\ \Gamma'\ (\lambda y.\ abs\ (f' y))) \\
\supset & (\forall \Gamma' \forall t'.\ \phi_{(y:\gamma)}(term)\ \Gamma'\ t' \supset S\ \Gamma'\ t')
\end{array}$$

For example it can be used with the invariant:

$$S := \lambda \Gamma' \lambda x'. \forall \Gamma \forall x.\ (\Gamma' = (\lambda y. \Gamma) \wedge x' = (\lambda y. x)) \supset term\ \Gamma\ x$$

After a similar sub-induction on $\phi_{(y:\gamma)}(mem)$ one will have obtained a derivation of:

$$\vdash \forall \Gamma \forall x.\ (\nabla y.\ term\ \Gamma\ x) \supset term\ \Gamma\ x$$

6.2.1 Proof theory of $\mu\mathbf{LJ}^{\nabla}$

The lifting transformation behaves nicely with respect to first-order abstraction, *i.e.,* $\phi_\sigma(F\sigma(t\sigma)) \equiv ((\lambda x'.\ \phi_\sigma(F\sigma(x'\sigma)))t)$. Unfortunately, the same does not hold for

second-order abstraction: $\phi_\sigma(B)\phi_\sigma(S)$ is not necessarily the same as $\phi_\sigma(BS)$. Consider an abstraction $B := (\lambda p \ldots \nabla\sigma'(\ldots p \ldots))$. In $\phi_\sigma(BS)$ the occurrence of p will become $\phi_{\sigma\sigma'}(S)$ whereas in $\phi_\sigma(B)\phi_\sigma(S)$ it becomes $\phi_{\sigma'}(\phi_\sigma(S))$.

This is technically complicating the metatheory, but our attempts to change the definition and avoid that have been unsuccessful. In fact, this permutation of names can be understood as inherent to the identification of $\nabla\sigma.\,\mu B$ and $\mu\phi_\sigma(B)$. Indeed, these two fixed points reveal two different prefixes of generic quantifiers after n unfoldings, respectively $\sigma(\sigma')^n$ and $(\sigma')^n\sigma$.

We shall establish, however, that the permutation of names does not affect provability. This can be derived in the logic itself, not only at the meta level. It will provide a way to bridge the gap between $\phi_\sigma(BS)$ and $\phi_\sigma(B)\phi_\sigma(S)$ by translating all permuted instances.

Proposition 6.4. *For any formula P, and any two generic contexts σ and σ^* permutations of each other, it is provable that* $(\nabla\sigma.\,P\sigma) \supset (\nabla\sigma^*.\,P\sigma)$.

The proof basically builds a corrected η-expansion, carrying the permutability from elementary formulas (equality, $\top$ and $\bot$) through all connectives including μ and ν. In a sense, there is nothing clever in it: at any point there is only one choice that keeps things well-typed. That said, it is a good test for the logic to check that everything goes as expected.

Proof. For any term abstraction F, for any σ and σ' permutations of each other, we denote by $[F]^{\sigma}_{\sigma'}$ the formula $\lambda x.\,F(\lambda\sigma'.x\sigma)$. In that proof let us refer to the original σ as σ_0. For any extension $\sigma_0\sigma'$ of the context σ_0 we shall denote by $(\sigma_0\sigma')^*$ the context $\sigma_0^*\sigma'$.

We shall prove the following generalization, by induction on B. Let (σ_i) be a family of extensions of σ_0, $(F_i)_i$ be a family of fixed points of types $(\sigma_i^* \to \gamma) \to o$, and σ be an extension of all the σ_i. Then we have:

$$(\lambda\vec{p'}.\,\phi^{\Gamma}_{\sigma}(B\sigma\vec{p}))([F_i]^{\sigma_i}_{\sigma_i^*})_i \vdash (\lambda\vec{p^*}.\,\phi^{\Gamma^*}_{\sigma^*}(B\sigma\vec{p}))(F_i)_i$$

Where $\Gamma := \langle p_1, \sigma_1, p_1'\rangle, \ldots, \langle p_n, \sigma_n, p_n'\rangle$ and $\Gamma^* := \langle p_1, \sigma_1^*, p_1^*\rangle, \ldots, \langle p_n, \sigma_n^*, p_n^*\rangle$, which implies that if p_i is of type $\gamma_i \to o$ then p_i' has type $(\sigma_i \to \gamma_i) \to o$ and p_i^* has type $(\sigma_i^* \to \gamma_i) \to o$.

- The cases of units $(\top, \bot)$ and covariant propositional connectives $(\wedge, \vee)$ can be easily checked. The case of $\supset$ relies on the symmetry of our goal, which allows us to conclude by induction hypothesis after a negation, with the respective roles of $\sigma_0, (\sigma_i)_i, \sigma, F_i$ and $\sigma_0^*, (\sigma_i^*)_i, \sigma^*, [F_i]^{\sigma_i}_{\sigma_i^*}$ being exchanged — notice in particular that $[[F]^{\sigma}_{\sigma^*}]^{\sigma^*}_{\sigma} \equiv F$. The case for equality consists in checking that $u = v \vdash (\lambda\sigma^*.u\sigma) = (\lambda\sigma^*.v\sigma)$.

- If B starts with a generic quantification then σ are extended, and we can conclude by induction hypothesis.

- For first-order quantification, the two cases are symmetric. We treat only the universal case, that is when B is of the form $(\lambda\sigma\lambda\vec{p}.\,\forall x.\,B'\sigma x\vec{p})$. For clarity we ignore the fixed point parameters, which do not interfere. Essentially,

we have to establish $\phi_\sigma(\forall x.\ F\sigma x) \vdash \phi_{\sigma^*}(\forall x.\ F\sigma x)$, that is $\forall x.\ \phi_\sigma(F\sigma(x\sigma)) \vdash \forall x.\ \phi_{\sigma^*}(F\sigma(x\sigma^*))$. This is done by introducing an universal variable h for the right hand-side universal quantification, and instantiating the left hand-side one by $(\lambda\sigma.\ h\sigma^*)$. The resulting goal $\phi_\sigma(F\sigma(h\sigma^*)) \vdash \phi_{\sigma^*}(F\sigma(h\sigma^*))$ is obtained by induction on $B := \lambda\sigma.F\sigma(h\sigma^*)$.

Notice that although we have made some *terms* larger when moving from $F\sigma x$ to $F\sigma(x\sigma)$, the *formula* structure of B is always kept intact, which is what matters in this induction.

- We now treat the case of the least fixed point, greatest fixed point being obtained in a symmetric way. For clarity we suppose that the fixed point has arity one, and leave out the other fixed point parameters of B which can be re-introduced without interfering at the cost of very heavy notations. We have to build a derivation of $\phi^\Gamma_\sigma(\mu(B'\sigma)(t\sigma)) \vdash \phi^{\Gamma^*}_{\sigma^*}(\mu(B'\sigma)(t\sigma))$, that is:

$$\mu(\lambda p'\lambda x.\ \phi^{\Gamma,\langle p,\sigma,p'\rangle}_\sigma(B'\sigma p(x\sigma)))\vec{t} \vdash \mu(\lambda p^*\lambda x.\ \phi^{\Gamma^*,\langle p,\sigma^*,p^*\rangle}_{\sigma^*}(B'\sigma p(x\sigma^*)))(\lambda\sigma^*.t\sigma)$$

We proceed by applying the induction rule with the right hand-side as the invariant:

$$S := [\mu(\lambda p^*\lambda x.\ \phi^{\Gamma^*,\langle p,\sigma^*,p^*\rangle}_{\sigma^*}(B'\sigma p(x\sigma^*)))]^\sigma_{\sigma^*}$$

One premise of the induction rule is thus an instance of the identity, the other is the proof of invariance, which conclusion must be:

$$(\lambda p'\lambda x.\ \phi^{\Gamma,\langle p,\sigma,p'\rangle}_\sigma(B'\sigma p(x\sigma)))S\,x \vdash S\,x$$

After an unfolding of μ on the right hand-side, we finally obtain a goal which can be obtained by induction hypothesis, with an extended family of F_i and σ_i:

$$(\lambda p'.\ \phi^{\Gamma,\langle p,\sigma,p'\rangle}_\sigma(B'\sigma p(x\sigma)))[\mu(\lambda p^*\lambda x.\ \phi^{\Gamma^*,\langle p,\sigma^*,p^*\rangle}_{\sigma^*}(B'\sigma p(x\sigma^*)))]^\sigma_{\sigma^*}$$

$$\vdash (\lambda p^*.\ \phi^{\Gamma^*,\langle p,\sigma^*,p^*\rangle}_{\sigma^*}(B'\sigma p(x\sigma)))(\mu(\lambda p^*\lambda x.\ \phi^{\Gamma^*,\langle p,\sigma^*,p^*\rangle}_{\sigma^*}(B'\sigma p(x\sigma^*))))$$

Recall that we are doing an induction on B: although we unfolded a fixed point, the process shall not go through the recursive occurrences of the fixed point (*e.g.,* the S substituted for p') but treats the predicate variables in B as a base case.

- If B is of the form $\lambda\sigma\vec{p}.\ p_i(t\sigma)$, then $\sigma = \sigma_i\sigma'$ and $\sigma^* = \sigma^*_i\sigma'$ for some σ', and we have:

$$\lambda\vec{p'}.\ \phi^\Gamma_\sigma(B\sigma\vec{p}) = \lambda\vec{p'}.\ \nabla\sigma'.p'_i(\lambda\sigma_i.t\sigma) \qquad \lambda\vec{p^*}.\ \phi^{\Gamma^*}_{\sigma^*}(B\sigma\vec{p}) = \lambda\vec{p^*}.\ \nabla\sigma'.p^*_i(\lambda\sigma^*_i.t\sigma)$$

Our goal is thus to build a derivation of:

$$\nabla\sigma'.\ [F_i]^{\sigma_i}_{\sigma^*_i}(\lambda\sigma_i.t\sigma) \vdash \nabla\sigma'.\ F_i(\lambda\sigma^*_i.t\sigma^*)$$

which is simply an instance of the identity:

$$\nabla\sigma'.\ F_i(\lambda\sigma^*_i.t\sigma) \vdash \nabla\sigma'.\ F_i(\lambda\sigma^*_i.t\sigma)$$

□

Corollary 6.5. *Proposition 6.4 actually provides a mean to transform a derivation into another one establishing the same sequent where some instances of $\phi_\sigma(F)$ have been replaced by some permutation $\phi_{\sigma^*}(F)$: this is done by cutting against η-expansions of the derivations provided by the proposition. In particular, it allows to compensate the difference between $\phi_\sigma(B)\phi_\sigma(S)$ and $\phi_\sigma(BS)$.*

Definition 6.6. We extend the notion of lifting to sequents:

$$\phi_\sigma\Big(x_1,\dots,x_n; P_1(x_i)_i,\dots,P_m(x_i)_i \vdash Q(x_i)_i\Big) :=$$
$$x'_1,\dots,x'_n; \phi_\sigma(P_1(x'_i\sigma)_i),\dots,\phi_\sigma(P_m(x'_i\sigma)_i) \vdash \phi_\sigma(Q(x'_i\sigma)_i)$$

Proposition 6.7 (Lifting derivations)**.** *For any σ, the provability of $\Sigma;\Gamma \vdash P$ implies that of $\phi_\sigma(\Sigma;\Gamma \vdash P)$*[1]*.*

This allows one to read a proof of $(\forall t.\ (\nabla a.\ p\ t) \supset p\ t)$ as not only establishing that the provability of p in *the context* of one unused generic variable entails that of p in the empty context, but more generally that the provability is stable by removal of an unused variable from *any* context. This is what makes our system expressive without any need for concrete manipulations of the context and other complex devices such as quantifications over all generic contexts.

Proof. By a simple induction on the structure of the proof, which is preserved by the transformation. We show here a couple key cases. Propositional cases are most trivial. Equality is trivial on the right, slightly less on the left, where

$$\frac{\{((x_i)_i; \Gamma(x_i)_i \vdash C(x_i)_i)\theta : \theta \in mgu(u(x_i)_i \doteq v(x_i)_i\}}{(x_i)_i; \Gamma(x_i)_i, u(x_i)_i = v(x_i)_i \vdash C(x_i)_i}$$

is transformed by induction hypothesis into

$$\frac{\{((x'_i)_i; \phi_\sigma(\Gamma(x'_i\sigma)_i) \vdash \phi_\sigma(C(x'_i\sigma)_i))\theta' : \theta' \in mgu((\lambda\sigma.u(x'_i\sigma)_i) \doteq (\lambda\sigma.v(x'_i\sigma)_i))\}}{(x'_i)_i; \phi_\sigma(\Gamma(x'_i\sigma)_i), (\lambda\sigma.u(x'_i\sigma)_i) = (\lambda\sigma.v(x'_i\sigma)_i) \vdash \phi_\sigma(C(x'_i\sigma)_i)}$$

The cases of first-order quantifications are straightforward:

$$\frac{(x_i)_i, x; \dots \vdash P(x_i)_i x}{(x_i)_i; \dots \vdash \forall x.\ P(x_i)_i x} \quad \rightsquigarrow \quad \frac{(x'_i)_i, x'; \dots \vdash \phi_\sigma(P(x'_i\sigma)_i x'\sigma)}{(x'_i)_i; \dots \vdash \forall x'.\ \phi_\sigma(P(x'_i\sigma)_i x'\sigma)}$$

$$\frac{(x_i)_i; \dots \vdash P(x_i)_i(t(x_i)_i)}{(x_i)_i; \dots \vdash \exists x.\ P(x_i)_i x} \quad \rightsquigarrow \quad \frac{(x'_i)_i; \dots \vdash \phi_\sigma(P(x'_i\sigma)_i(t(x'_i\sigma)_i))}{(x'_i)_i; \dots \vdash \exists x'.\ \phi_\sigma(P(x'_i\sigma)_i(x'\sigma))}\ x' := \lambda\sigma.t(x'_i\sigma)_i$$

Finally, thanks to Corollary 6.5, fixed point rules can also be translated into lifted instances of themselves, plus some corrective cuts. □

[1] We do not follow the notational conventions here: of course, the variables of Σ can occur in Γ and P, unlike those of σ.

Proposition 6.8 (Conservativity & expressiveness)**.** *We call* 0*-provability the provability without any use of the (co)induction rules. Let P be a formula, possibly involving* ∇ *quantifications.*

1. *The* 0*-provability of P in* μLJ^{∇_0} *is equivalent to its* 0*-provability in* μLJ^{∇}*.*
2. *Moreover, the provability of P in* μLJ^{∇_0} *implies its provability in* μLJ^{∇}*,*
3. *but the converse is false.*

Proof. Each direction of (1) is done by induction on the 0-derivation. Both are straightforward proof transformation similar to those detailed before, including corrective cuts (*cf.* Corollary 6.5) as in Proposition 6.7. For (2) we add the translation of an induction in $\mu\mathrm{LJ}^{\nabla_0}$ to $\mu\mathrm{LJ}^{\nabla}$, which amounts to lift the invariant and the invariance proof. Finally, (3) shall be seen later, with the ability to weaken the generic context in some cases in $\mu\mathrm{LJ}^{\nabla}$, which is impossible in $\mu\mathrm{LJ}^{\nabla_0}$. □

6.2.2 Cut-elimination

We adapt the proof reductions involved in cut-elimination, and argue that the termination is not affected, leaving a detailed proof of that for future work. The only novelty is the transformation ϕ. It only affects second-order instantiations in fixed point unfoldings, induction and coinduction. It does not affect several important properties of the system: proofs can be instantiated, the signature can be enriched, etc. The non-trivial part is adapting the reduction for eliminating a cut on a fixed point. We only show the case of the least fixed point, the greatest being similar.

The essential reduction for least fixed point is the following:

$$\dfrac{\dfrac{\Pi_S}{BS\,\vec{x} \vdash S\,\vec{x}} \quad \dfrac{\dfrac{\Pi'}{\Gamma, S\,\vec{t} \vdash P}}{\Gamma, \mu B\vec{t} \vdash P} \quad \dfrac{\Pi}{\Gamma \vdash \mu B\vec{t}}}{\Gamma \vdash P} \longrightarrow \dfrac{\dfrac{\Pi'}{\Gamma, S\,\vec{t} \vdash P} \quad \dfrac{fold(\Pi, \Pi_S)}{\Gamma \vdash S\,\vec{t}}}{\Gamma \vdash P}$$

Where the $fold(\Pi, \Pi_S)$ transformation replaces in Π all unfoldings of μB by a cut against Π_S. More precisely, since the unfoldings might be lifted, occurrences of $\phi_\sigma(\mu B) = \mu(\phi_\sigma(B))$ are replaced by $\phi_\sigma(S)$:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \phi_\sigma(B)(\mu(\phi_\sigma(B)))\vec{t}}}{\Gamma \vdash \mu(\phi_\sigma(B))\vec{t}}\ \mu R \longrightarrow \dfrac{\dfrac{\vdots}{\phi_\sigma(B)\phi_\sigma(S)\vec{t} \vdash \phi_\sigma(S)\vec{t}} \quad \Gamma \vdash \phi_\sigma(B)\phi_\sigma(S)\vec{t}}{\Gamma \vdash \phi_\sigma(S)\vec{t}}\ cut$$

To complete this, one must build from Π_S, for a given σ and $\vec{t}$, a proof of $\phi_\sigma(B)\phi_\sigma(S)\vec{t} \vdash \phi_\sigma(S)\vec{t}$. Using Proposition 6.7 on Π_S we get a derivation of $\phi_\sigma(BS\,\vec{x}) \vdash \phi_\sigma(S\,\vec{x})$. Then, Corollary 6.5 gives a proof of $\phi_\sigma(B)\phi_\sigma(S)\vec{x} \vdash \phi_\sigma(S)\vec{x}$ where $\vec{x}$ can finally be instantiated by $\vec{t}$.

We leave the termination of the reduction for further work. Strictly speaking, cut-elimination has not been established even for $\mu\mathrm{LJ}^{\nabla_0}$, which does not have the stratification constraints on which is built the proof for LINC [Tiu04]. However, based on

our work on μMALL, we believe that it holds under the simple constraint of monotonicity. But even with the stratification constraint on fixed points, the termination of cut-elimination for LINC does not carry easily to $\mu\mathrm{LJ}^{\nabla}$, because of the extra cuts inserted in the above reduction for translating between $\phi(BS)$ and $\phi(B)\phi(S)$.

6.2.3 Structural rules on the generic context

Minimal generic quantification is characterized by the absence of strengthening or exchange on generic contexts. This has not been changed with $\mu\mathrm{LJ}^{\nabla}$. For example, $(\nabla_\gamma x.\ \top) \supset (\exists_\gamma x.\ \top)$ still cannot be derived without assuming the non-vacuity of γ, which would make even $(\exists_\gamma x.\ \top)$ alone derivable. Symmetrically, $\forall$ does not imply ∇ in general. And vacuous generic quantifications cannot *a priori* be added or removed.

We show, however, that the missing generic strengthening and weakening are actually derivable for a reasonable class of formulas. On such formulas, the minimality does not make generic quantification weaker than other approaches, just as in the finite behavior case. The essential idea for obtaining generic strengthening, that is $(\nabla x.\ P) \supset P$, is to forbid positive occurrences of existential quantification, unless they are guarded by a formula that ensures that the existential variable does not depend on the extra generic variable x.

Definition 6.9. A *guard* is a formula G such that for any σ:

$$\forall y'.\ \phi_{x\sigma}(G\sigma(y'x\sigma)) \supset \exists y.\ y' = (\lambda x.\ y) \wedge \phi_\sigma(G\sigma(y\sigma))$$

A typical guard would be an equality $(\lambda\sigma\lambda y.\ u\sigma y = v\sigma y)$ such that all unifiers of $(\lambda x\sigma.\ u\sigma(y'x\sigma) = v\sigma(y'x\sigma))$ set $y' := \lambda x\lambda\sigma.y\sigma$ for some y. This holds for equalities found in most fixed point definitions, which fully define newly introduced existential variables as compounds or sub-terms of pre-existing terms.

Definition 6.10. The fragments $\mathcal{W}$ and $\mathcal{S}$ (respectively standing for $\mathcal{W}$eakening and $\mathcal{S}$trengthening) are mutually defined by the following grammar, where $\mathcal{G}$ denotes a guard:

$$\begin{array}{rcl}
\mathcal{W} & ::= & \mathcal{W} \wedge \mathcal{W} \mid \mathcal{W} \vee \mathcal{W} \mid \top \mid \bot \mid u = v \mid \nabla x.\mathcal{W} \mid \mu\mathcal{W} \mid \nu\mathcal{W} \\
& \mid & \mathcal{S} \supset \mathcal{W} \mid \exists x.\ W \mid \forall x.\ \mathcal{G}x \supset \mathcal{W} \\
\mathcal{S} & ::= & \mathcal{S} \wedge \mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid \top \mid \bot \mid u = v \mid \nabla x.\mathcal{S} \mid \mu\mathcal{S} \mid \nu\mathcal{S} \\
& \mid & \mathcal{W} \supset \mathcal{S} \mid \forall x.\ \mathcal{S} \mid \exists x.\ \mathcal{G}x \wedge \mathcal{S}
\end{array}$$

Proposition 6.11. *For any formula $W \in \mathcal{W}$ (resp. $S \in \mathcal{S}$) it is provable that $W \supset \nabla x.\ W$ (resp. $\nabla x.\ S \supset S$).*

An other way to put it, for example, is that the following rules are admissible in $\mu\mathrm{LJ}^{\nabla}$:

$$\frac{\Gamma, \sigma, \sigma' \rhd S\sigma\sigma' \vdash G}{\Gamma, \sigma, x, \sigma' \rhd S\sigma\sigma' \vdash G} \qquad \frac{\Gamma \vdash \sigma, \sigma' \rhd W\sigma\sigma'}{\Gamma \vdash \sigma, x, \sigma' \rhd W\sigma\sigma'}$$

Proof. Let α be the type of x, the unused generic variable. We denote by $[F]^{\uparrow}$ the formula $\lambda x'.\ \forall x.\ x' = (\lambda\_.\ x) \supset F\ x$. If F has type $\gamma \to o$, then $[F]$ has type $(\alpha \to \gamma) \to$

o. Conversely, we denote by $[F']^{\downarrow}$ the formula $\lambda x.\, F'(\lambda\_.\, x)$. These conversions satisfy the following equivalences:

$$\forall x. \quad Fx \equiv [F]^{\uparrow}(\lambda\_.\, x) \quad \wedge \quad F'(\lambda\_.\, x) \equiv [F']^{\downarrow}x$$

We establish by simultaneous induction on $\mathcal{S}$ and $\mathcal{W}$, for any generic context σ and any families of fixed points $(F_i)_i$ of types $((\sigma_i \to \gamma_i) \to o)_i$ and $(F'_j)_j$ of types $((\alpha\zeta_j \to \delta_j) \to o)_j$, that for $B \in \mathcal{S}$:

$$(\lambda\vec{p'}\lambda\vec{q'}.\, \phi^{\Gamma'}_{x:\alpha,\sigma}(B\sigma\vec{p}\vec{q}))([F_i]^{\uparrow})_i(F'_j)_j \vdash (\lambda\vec{p'}\lambda\vec{q'}.\, \phi^{\Gamma}_{\sigma}(B\sigma\vec{p}\vec{q}))(F_i)_i([F'_j]^{\downarrow})_j$$

and for $B \in \mathcal{W}$:

$$(\lambda\vec{p'}\lambda\vec{q'}.\, \phi^{\Gamma}_{\sigma}(B\sigma\vec{p}\vec{q}))(F_i)_i([F'_j]^{\downarrow})_j \vdash (\lambda p''\lambda q''.\, \phi^{\Gamma'}_{x:\alpha,\sigma}(B\sigma\vec{p'}\vec{q'}))([F_i]^{\uparrow})_i(F'_j)_j$$

Where $\Gamma := (\langle p_i, \sigma_i, p'_i\rangle)_i, (\langle q_j, \zeta_j, q'_j\rangle)_j$ and $\Gamma' := (\langle p_i, \alpha\sigma_i, p''_i\rangle)_i, (\langle q_j, \alpha\zeta_j, q''_j\rangle)_j$.

The two inductive processes of weakening and strengthening are interleaved, the switching from one to the other being done when treating the negation involved in $\supset$. When encountering a least fixed point on the left, an induction step will be done, adding on the left a corrected parameter ($[\bullet]^{\uparrow}$ or $[\bullet]^{\downarrow}$) and its bare version on the right. With greatest fixed points, the corrected version appears on the right. When in the strengthening (resp. weakening) phase, the new parameters will be fixed points over strengthenable (resp. weakenable) bodies $\nu\mathcal{S}$ or $\mu\mathcal{S}$ (resp. $\nu\mathcal{W}$ or $\mu\mathcal{W}$). By monotonicity, only strengthenable parameters will be used in the strengthening phase in the base case where B is a projection, *e.g.,* $\lambda\vec{p}\lambda\vec{q}.p_i(t\sigma)$. However, this is not exploited in the proof, allowing us to mix weakenable and strengthenable parameters, thus having two instead of four families of parameters.

Having outlined the proof, we now only detail a few key steps of strengthening:

- For strengthening, universal quantification is harmless. We basically have to derive:
$$\forall x''.\, \phi_{x\sigma}(B\sigma(x''x\sigma)) \vdash \forall x'.\, \phi_{\sigma}(B\sigma(x'\sigma))$$
After introducing the universal variable $x' : \sigma \to \gamma$ on the right we instantiate the universal on the left with $(\lambda\_\lambda\sigma.x''\sigma)$.

- The case of existential quantification crucially relies on guardedness, but is straightforward by definition of the guard. An universal variable $x'' : \alpha\sigma \to \gamma$ is introduced on the left, together with a guard hypothesis. After a cut against the formula provided by Definition 6.9, the unification of the left equality restricts x'' to be some $(\lambda\_.\, x')$ and provides a proof of the guard for x'. This is all we need to instantiate the right hand-side existential, prove the guard, and conclude by induction hypothesis.

- Suppose $B = \lambda\sigma\lambda\vec{p}\lambda\vec{q}.\, p_i(t\sigma)$, with σ of the form $\sigma_i\sigma'$. Then $\phi^{\Gamma'}_{x:\alpha,\sigma}(B\sigma\vec{p}\vec{q})$ is $\nabla\sigma'.\, p''_i(\lambda\_\sigma_i.\, t\sigma_i\sigma')$ and $\phi^{\Gamma}_{\sigma}(B\sigma\vec{p}\vec{q})$ is $\nabla\sigma'.\, p'_i(\lambda\sigma_i.\, t\sigma_i\sigma')$. Thus, we have to prove the following, which is actually an equivalence as remarked above:
$$\nabla\sigma'.\, [F_i]^{\uparrow}(\lambda\_\sigma_i.\, t\sigma_i\sigma') \vdash \nabla\sigma'.\, F_i(\lambda\sigma_i.\, t\sigma_i\sigma')$$

- If B is a least fixed point $\lambda\sigma\lambda\vec{p}\lambda\vec{q}.\ \mu(B'\sigma\vec{p}\vec{q})(t\sigma)$, then we apply the induction rule with the invariant $[S]^{\uparrow}$ where

$$S := \mu(\lambda X'\lambda x.\ (\lambda\vec{p'}\lambda\vec{q'}.\ \phi_{\sigma'}^{\Gamma,\langle X,\sigma,X'\rangle}(B'\sigma\vec{p}\vec{q}Xx))(F_i)_i([F'_j]^{\downarrow})_j)$$

  The main premise of the induction rule is trivial, of the form $[S]^{\uparrow}(\lambda\_.\ t) \vdash S\,t$. In the invariance premise (roughly of the form $\phi(B)[S]^{\uparrow}x' \vdash [S]^{\uparrow}x'$), the universal and implications resulting from the conversion from S to $[S]^{\uparrow}$ are introduced, and the left rule for equality changes the universal variable x' into some vacuous abstraction $(\lambda\_.\ x)$. After a right unfolding the resulting sequent can be derived by induction, with an extended family of $(F_i)_i$.

□

The previous proposition is very useful in practice. Indeed, most common fixed points (*nat*, *append*, *typeof*, etc.) are both in $\mathcal{S}$ and $\mathcal{W}$: they do not involve any universal quantification and the existential quantifications are guarded by equalities. Most of the time, the existentials are actually very weak in that they do not even require any invention, such as in $\exists y.\ x = s\ y \wedge \ldots$.

Another example of guard would be *nat* itself, as it forces its parameter to be fully defined in terms of the constants zero and successor. We let the reader check that the following can be derived in $\mu\mathrm{LJ}^{\nabla}$: $\nabla x.\ nat(y'x) \supset \exists y.\ y' = (\lambda x.y) \wedge nat\ y$. More interestingly, it should be possible to characterize a fragment of valid guards, which could build on top of guards like our $\mathcal{W}$ and $\mathcal{S}$ did, such that from a basic guard (*e.g.,* $\lambda y.\ x = s\ y$) one could derive an other (*e.g., nat*), and an other (*e.g., natlist*), etc.

Example 6.12. The typical example of a fixed point that does not fall in $\mathcal{S}$ is provability in a first-order logic. Some of the key clauses of its specification would be:

$$\begin{aligned}
prove\ \Gamma\ P &\triangleq P \in \gamma \\
prove\ \Gamma\ (P \hat{\supset} Q) &\triangleq prove\ (P :: \Gamma)\ Q \\
prove\ \Gamma\ (\hat{\forall}(\lambda x.\ Px)) &\triangleq \nabla x.\ prove\ \Gamma\ (Px) \\
prove\ \Gamma\ (\hat{\exists}(\lambda x.\ Px)) &\triangleq \exists x.\ prove\ \Gamma\ (Px)
\end{aligned}$$

With minimal generic quantification, the generic context exactly represents the signature of the object sequent. The fixed point *prove* is not in $\mathcal{S}$ because the last clause involves an existential quantification that is not guarded at all, and can notably range over variables present in the object-level signature (that is the generic context) even though not anywhere in Γ and P. Indeed, it does not always hold that $prove\ \Gamma\ (\hat{\forall}(\lambda x.\ P)) \supset prove\ \Gamma\ P$, which is essentially a generic strengthening on $prove\ \Gamma\ P$. For example, $\forall x.\ \exists y.\ \top$ is provable in the object logic, but $\exists y.\ \top$ is not unless we assume the existence of a term.

6.3 Practical use of μLJ^∇

We describe here a few significant examples of what can be done with μLJ^∇. These examples have been checked[2] using the interactive theorem prover Taci [SBM07], which is described in more details in the next chapter. Starting with the initial implementation of μLJ in Taci, we added support for the transformation ϕ, thereby obtaining a convenient implementation of μLJ^∇. Formulas are converted to ϕ-normal form as late as possible, since it is more readable to display generic variables in an explicit generic context. The ϕ-normal form is used, for example, by the automated inductive proof-search strategy implemented in Taci. This automatically extends the tactic from μLJ to μLJ^∇. That basic amount of automation relieves the user from proving bureaucratic lemmas such as strengthening of the generic context, which alleviates the cost of working with minimal generic quantification instead of a stronger variant.

6.3.1 The copy program

In this example we shall work on a representation of untyped λ-terms. We assume a signature with a type *tm* and two constants: *app* of type $tm \to tm \to tm$ and *abs* of type $(tm \to tm) \to tm$. Notice that since these are term-level constants and not fixed point definitions, there is nothing wrong with negative occurrence of *tm* in the type of *abs*. The `copy` program is defined as follows in λProlog:

```
copy (app M N) (app P Q) := copy M P, copy N Q.
copy (abs M) (abs P) :=
    pi x\ pi y\ copy x y -> copy (M x) (N y).
```

It can be used for example to substitute a term into another: `copy A B -> copy M N` requires that `N` is `M` where some (but not necessarily all) occurrences of `A` are changed into `B`.

There are mainly two approaches for encoding such a program in our logic. The first one is essentially the two-level approach [MM02] taken for example in the Abella system [Gac08]. It consists in specifying λProlog proof-search as an object logic, and reasoning about its behavior on the `copy` program. Given the initial formulation of the problem, this is best from an adequacy point of view, but it is heavy and notably does not allow direct inductions on the structure of `copy`. Instead, we encode the program directly. We argue that this preserves most essential points, and is adequate.

The encoding of the universal quantification in the abstraction clause is a ∇ quantification, reflecting the introduction of a generic variable. In order to encode in a monotonic way the implication in that `abs` clause, we introduce a context parameter representing the `copy` atoms present in the λProlog context. In the encoding, a new clause appears, stating that if `copy M N` is found in the context, then it holds.

$$\begin{array}{rcl} copy & := & \mu\, copy.\, \lambda\Gamma MN. \\ & & \langle M, N\rangle \in \Gamma \end{array}$$

[2] However, the code shown in this section is often not valid for Taci, but has been simplified for readability.

$$\begin{array}{ll} \vee & (\exists M_1 M_2 N_1 N_2.\ M = (app\ M_1\ M_2) \wedge N = (app\ N_1\ N_2) \wedge \\ & \quad copy\ \Gamma\ M_1\ N_1 \wedge copy\ \Gamma\ M_2\ N_2) \\ \vee & (\exists M_1 N_1.\ M = (abs\ M_1) \wedge N = (abs\ N_1) \wedge \\ & \quad \nabla xy.\ copy\ (\langle x, y\rangle :: \Gamma)\ (M_1 x)\ (N_1 y)) \end{array}$$

We shall prove an useful fact about that inductive definition:

$$\forall MN.\ copy\ []\ M\ N \supset M = N$$

This property should be proved by induction over *copy*. A naive invariant would be that the context contains only pairs (m, m). It does not hold: when going under an abstraction, two generic variables are introduced, marked equal in the context Γ. Since they are generic, it does not break the result, but does make the invariant more complex.

In fact, we proceed by proving that *copy* implies *eq*, where *eq* is defined as a least fixed point mostly like *copy* except for its abstraction clause:

$$eq\ (abs\ M)\ (abs\ N) := \nabla x.\ eq\ (\langle x, x\rangle :: \Gamma)\ (M\ x)\ (N\ x)$$

For *eq*, it is now easy to show that if the context contains only pairs (m, m), it will remain true and hence *eq* implies equality. It remains to show that *copy* implies *eq*, which actually holds for any Γ. This is done by induction over *copy*, the interesting case being that of abstraction:

$$\begin{array}{ll} \forall \Gamma MN. & (\nabla ab.\ eq\ (\langle a, b\rangle :: \Gamma)\ (M\ a)\ (N\ b)) \\ \supset & (\nabla a\ .\ eq\ (\langle a, a\rangle :: \Gamma)\ (M\ a)\ (N\ a)) \end{array}$$

This goal really expresses the heart of our problem with the shape of the context. It requires an induction under two generic quantifications. Basically, the invariant should state that two generic variables can be merged. After having lifted *eq* over the generic quantifications on *a* and *b*, this can actually be written elegantly as a simple invariant, and the proof of invariance is straightforward:

$$\lambda \Gamma'' M'' N''.\ \nabla a.\ eq\ (\Gamma''\ a\ a)\ (M''\ a\ a)\ (N''\ a\ a)$$

We do not know of any other system where it is possible to obtain that name merging principle in such a direct way. The proofs cited above can however be carried out in other logics, for example in LG [Tiu06] as pointed out by Gacek, thanks to the strengthening on generic variables. Since that principle is admissible in μLJ^{∇} for the particular case of *copy* and *eq*, that observation gives us an alternative, less concise proof.

6.3.2 λ-calculus

We now discuss the specification of simply typed, Church-style λ-calculus, and the proofs of subject reduction and determinacy of typing. The signature for terms will consist of two types: *ty* for simple types and *tm* for λ-terms; two constants for terms: $app : tm \to tm \to tm$ and $lambda : ty \to (tm \to tm) \to tm$; the constant $arrow : ty \to ty \to ty$ for types, as well as some arbitrary base types.

We shall not detail the definition of a predicate `bind` such that when Γ is a list of pairs representing bindings, `bind` Γ k v expresses that $\langle k, v\rangle \in \Gamma$. We define a least fixed point `typeof` such that the typing judgment ($\Gamma \vdash_{\Lambda^{\to}} m : t$) is represented by (`typeof` Γ m t):

```
inductive typeof G M T :=
  (bind G M T) ;
  (sigma t\m1\m2\
     M = (app m1 m2), typeof G m1 (arrow t T), typeof G m2 t) ;
  (sigma t\t'\f\
     M = lambda t f, T = arrow t t',
     nabla x\ typeof (cons (pair x t) G) (f x) t').
```

Along these lines, we also specify one-step β-reduction as a least fixed point called `one`, and prove subject reduction as well as determinacy of typing. However, the statement of the theorems required particular care. In this style of specification, a variable is nothing but a placeholder for a term. For example, nothing forbids the typing context Γ to contain constructed terms, instead of only variables as usual. This seems interesting, but certainly differs from the informal practice. One can try to stick to the usual notion of context; for example, we established subject reduction under the assumption that the context does not contain constructed terms:

```
theorem subject_reduction :
  pi m\n\ one m n =>
   pi G\
    (pi t\a\t'\ bind G (lambda t a) t' => false) =>
    (pi a\b\t'\ bind G (app a b) t' => false) =>
    pi t\ typeof G m t => typeof G n t.
```

The corresponding proof of subject reduction in $\mu\mathrm{LJ}^{\nabla}$ could not be completely built within Taci: a small gap is left because our tool currently strictly supports only higher-order pattern unification. In any case, this is only a problem of the implementation, not of the logic.

For typed determinacy, we used an alternative notion of context. Instead of assuming that keys are unique and not constructed, we assumed that each binding satisfied type determinacy:

```
context G := pi x\t\ bind G x t => pi t'\ typeof G x t' => t=t'.
theorem type_determinacy :
  pi g\x\t\ typeof g x t => context g =>
            pi t'\ typeof g x t' => t=t'.
```

This formulation implies new branches in the proof compared to the informal one. However, they are trivially treated, and overall the proof is as simple as expected from the informal practice. Moreover, the resulting theorem is also slightly stronger than the usual one, as it allows richer contexts.

6.4 Related work

There are other approaches to reasoning on specifications involving variable binding. The *nominal* approach [Pit03] is popular, notably because of its good support in Isabelle with the nominal package [UT05]. However, it is not a proof-theoretic but an axiomatic solution: variables are treated as names, a special kind of objects about which some axioms are postulated, and some machinery is developed to automate the definition and seamless manipulation of notions such as substitution or freshness for data-types involving binding.

Several systems have been designed to manipulate HOAS specifications. We have already evoked Twelf's metatheorem prover, based on the meta-logic $\mathcal{M}_2$ [SP98], a restricted logic whose terms are LF objects. There are various attempts to develop programming languages supporting binders in data-types, notably Delphin [PS08] and Beluga [Pie08]. The latter system seems closely related in spirit to minimal generic quantification, since it keeps track of contexts in a precise and rigid way. But there are also some differences[3], and it remains difficult to relate the two systems.

The logics most closely related to $\mu\mathrm{LJ}^\nabla$ are certainly LG [Tiu06] and $\mathcal{G}$ [GMN08]. LG was designed by Tiu to solve the expressiveness problems of LINC. To obtain LG, he essentially added strengthening, weakening and exchange to generic contexts, hence obtaining $\forall x.\ Px \supset \nabla x.\ Px$ and $\nabla x.\ Px \supset \exists x.\ Px$. Although it might be practical, this does not answer the initial proof-theoretic question, since it changes the meaning of ∇. Moreover, it should be noted that the stronger semantic of generic quantification does not match the needs of some specifications. Recall Example 6.12: with the same specification of *prove* in LG we would not be representing the same object logic as in $\mu\mathrm{LJ}^\nabla$, but a more liberal one where $\exists x.\top$ is always provable. By admitting generic weakening and strengthening in LG, one implicitly assumes infinitely many available generic variables, and this assumption colors the object logic specified in LG. This inadequacy can be seen as analogous to the impossibility to represent directly a linear logic in an intuitionistic framework.

In fact, LG still lacks in expressiveness. For example, it seems impossible to derive $\forall l \forall x'.\ (\nabla n.\ \mathit{mem}\ (x'\ n)\ l) \supset \exists x.\ x' = (\lambda n.\ x)$ in that logic. It would be interesting to study the possibility of solving this issue by applying the lifting idea that is behind $\mu\mathrm{LJ}^\nabla$. Following another approach, Gacek et al. [GMN08] have developed the logic $\mathcal{G}$ as an extension of LG that retrieves enough expressiveness. $\mathcal{G}$ extends the notion of definition by allowing ∇ quantifications in the head of clauses[4], which allows to express properties like freshness, being a name, and generally brings enough expressiveness to handle large examples. This is put into practice in the theorem prover Abella [Gac08], which implements the logic $\mathcal{G}$.

[3] In $\mu\mathrm{LJ}^\nabla$, $\mathit{nat}\ x$ implies that x is some $s^n 0$, whatever the generic context. Conversely, $(\nabla x.\ \mathit{nat}\ x)$ is absurd. But in Beluga, any type can be open: sx is a valid natural number in a context containing $x : \mathit{nat}$.

[4] By the way, the resulting notion of definition cannot be encoded as fixed points anymore.

6.5 Conclusion

Thanks to a reformulation of the ∇ quantifier as a defined connective, we have revealed the expressiveness of minimal generic quantification in presence of fixed points. This resulted in the logic $\mu\mathrm{LJ}^{\nabla}$, which is a good system for writing specifications in the HOAS style, and reasoning in an expressive way about them. Satisfyingly, $\mu\mathrm{LJ}^{\nabla}$ seems to be a natural logic to consider, since its inference rules, the same as μLJ, are standard.

We have shown the implications, in terms of adequacy, of using logics with a more liberal treatment of generic variables, *i.e.,* admitting generic weakening and strengthening. And we have seen that the recovered expressiveness of $\mu\mathrm{LJ}^{\nabla}$ allows to derive these principles for important classes of formulas. There is no reason to doubt that it extends beyond these classes, whenever a principle is valid with respect to the minimal semantic of ∇. It should also be possible to build other classes corresponding to other common logical principles, such as $(\nabla x.\, Px) \supset (\forall x.\, Px)$, or more exotic ones like $(\nabla xy.\, Pxy) \supset (\nabla x.\, Pxx)$. More practically, such results should be integrated in mechanized theorem provers to simplify their use. A strength of $\mu\mathrm{LJ}^{\nabla}$ in that respect is that it is a very mild extension of μLJ, hence results and heuristics for μLJ easily extend to it. This has been verified in the proof assistant Taci, where the automated tactic designed for μLJ became, after straightforward modifications, an helpful one for $\mu\mathrm{LJ}^{\nabla_0}$.

From a theoretical point of view, we expect that the essential ideas behind the design of $\mu\mathrm{LJ}^{\nabla}$ could be put to work in other settings. It remains, however, to fully develop the proof-theoretical study of $\mu\mathrm{LJ}^{\nabla}$, notably concerning the termination of cut-elimination. First-order connectives usually play a negligible role in the complexity of cut-elimination; we hope that a different presentation of the essential idea behind $\mu\mathrm{LJ}^{\nabla}$ could make this true for ∇.

Chapter 7

Implementations

This thesis would be incomplete without an overview of the tools that we developed during its course. Not only do these implementations concretely establish the applicability of the work presented before, but they were actually a key part in designing and understanding our systems, notably μMALL, μLJ and their μ-focusing, but also our treatment of generic quantification in μLJ^{∇}.

We shall present two tools in that chapter: Bedwyr and Taci. Bedwyr [BGM+07a, BGM+07b] is a logic programming language mostly implementing ideas from earlier work on definitions and generic quantification [TNM05], but which fits nicely in our study of focusing for fixed points. Taci [BSV08] is an interactive theorem prover for μLJ^{∇}, developed as a framework for testing automated proof-search ideas, in particular applying our μ-focusing to inductive theorem proving.

7.1 Bedwyr

Bedwyr — Exactly. So, logically...
Villager — If... she... weighs... the same as a duck, ... she's made of wood.
Bedwyr — And therefore?
Villager — A witch!

Monthy Pythons, *The Holy Grail*, Scene 5

We have seen in Chapter 2 that the notion of fixed point gives an account of usual logic programming, but also supports case analysis. Building on that idea, Bedwyr offers a generalization of logic programming that allows model checking directly on syntactic expressions. Moreover, Bedwyr supports the *λ-tree* approach [Mil00] to higher-order abstract syntax [MN87, PE88], using term-level λ-binders and generic quantification.

7.1.1 Architecture

Bedwyr essentially builds focused derivations in μLJL (*cf.* Section 4.2), *i.e.,*allowing only synchronous connectives on the left, but considering both kinds of fixed points

as synchronous. Bedwyr only unfolds fixed points, and never uses the axiom rule, so it is not shocking that least and greatest fixed points are treated equally. This simple fragment is enough for expressing finite failure logically, and more generally capture simple model checking problems as well as may and must behavior in operational semantics.

We restrict ourselves to sequents whose goal lies in the fragment $\mathcal{G}$, and hypothesis in $\mathcal{H}$:

$$\begin{array}{rcl} \mathcal{G} & ::= & \mathcal{G} \wedge \mathcal{G} \mid \mathcal{G} \vee \mathcal{G} \mid s = t \mid \exists x.\mathcal{G}x \mid \mu(\lambda p \lambda \vec{x}.\mathcal{G})\vec{t} \mid \nu(\lambda p \lambda \vec{x}.\mathcal{G})\vec{t} \mid p\vec{t} \\ & \mid & \forall x.\mathcal{G}x \mid \mathcal{H} \supset \mathcal{G} \\ \mathcal{H} & ::= & \mathcal{H} \wedge \mathcal{H} \mid \mathcal{H} \vee \mathcal{H} \mid s = t \mid \exists x.\mathcal{H}x \mid \mu(\lambda p \lambda \vec{x}.\mathcal{H})\vec{t} \mid \nu(\lambda p \lambda \vec{x}.\mathcal{H})\vec{t} \mid p\vec{t} \end{array}$$

In this fragment, *all* the left rules are invertible. Consequently, we use a simple proof strategy that alternates between left and right rules, with the left rules taking precedence over the right rules. The resulting proofs correspond to reasoning by exhaustive case analysis, and never involve the axiom rule. The focusing system for μLJL reflects this strategy, but precises that completeness requires the ability to freeze fixed points. Indeed, Bedwyr only finds proofs that never freeze a fixed point, and hence never use the axiom rule.

This fragment, and the associated proof-search strategy, was first identified in LINC [Tiu04] as the Level-0/1 fragment [TNM05]. We still refer to $\mathcal{G}$ formulas as *Level-1* formulas, and $\mathcal{H}$ as Level-0 formulas, since it corresponds well to their operational proof-search treatment.

Two provers

Level-0 formulas are essentially a subset of goal formulas in λProlog, with ∇ replacing $\forall$. Proof search for a goal of Level-0 is thus the same as in λProlog, and Bedwyr implements that fragment following the basic ideas described in [EP91].

The novelty in Bedwyr is the support of implication in Level-1 formulas. Its operational treatment relies on the symmetries of sequent calculus: a derivation by exhaustive case analysis of $\forall x.\, Px \vdash Qx$ consists in derivations of $\forall \vec{y}.\, Q(t\vec{y})$ for each $(t\vec{y})$ such that $\vdash P(t\vec{y})$ is provable. A subtle aspect of this is the dual roles of existential and universal variables, reflected by our treatment of equality. For example:

$$\dfrac{\dfrac{\dfrac{}{\vdash s0 = s0}}{\vdash \exists z.\, s0 = sz}}{x; x = s0 \vdash \exists z.\, x = sz} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y; \vdash sy = sy}}{y; \vdash \exists z.\, sy = sz}}{x, y; x = sy \vdash \exists z.\, x = sz}}{x, y; x = sy \vdash \exists z.\, x = sz}}{x; \exists y.\, x = sy \vdash \exists z.\, x = sz}$$

$$x; x = s0 \vee \exists y.\, x = sy \vdash \exists z.\, x = sz$$

To reflect this operational reading of implications, the Level-1 prover is the same usual depth-first goal-directed prover as for Level-0, except for the treatment of implication. When the Level-1 prover reaches an implication $P \supset Q$, it calls the Level-0 prover on P and gets in return a collection of answer substitutions: the Level-1 prover

then checks that, for every substitution θ in that collection, $Q\theta$ holds. In particular, if Level-0 finitely fails with P, the implication is proved.

During proof search, existential and universal variables arise — we can ignore generic quantification for now. In the Level-1 prover, existential variables, introduced by the $\exists$ quantifier, are the usual instantiatable *logic variables*; universal variables are scoped constants, introduced by $\forall$. But in Level-0, $\exists$ introduces universal variables, which are instantiated by the left equality rule. Unfortunately, proof-theory does not give an account of existential variables: in proof-theory, $\exists$ does not introduce a variable but is instantiated by a term. In particular, the observation that underlies our treatment of implication does not explain how to treat existential variables. It is in fact a difficult problem, of a very different nature, that involves *disunification*. Consider, for example, proving $\exists x.\ x = 0 \supset x = 0$: despite the obvious solution $x := 0$, there may be infinitely many others, such as $x := 1$, which make the left equality absurd. Consequently, we reject that situation: if a logic variable occurs in Level-0, the system issues an error.

As with most depth-first implementations of proof search, Bedwyr suffers from other aspects of incompleteness: for example, the prover can easily loop during a search although different choices of goal or clause ordering can lead to a proof, and certain kinds of unification problems should be delayed instead of attempted eagerly. For a more detailed account on these incompleteness issues, we refer the reader to [BGM+06].

Generic quantification

As explained in Chapter 6, generic quantification is a rather orthogonal extension of sequent calculus. The same is true about its implementation in Bedwyr: the provers simply have to keep track of the generic context, in order to lift introduced variables over it, and treat the generic context as a λ-abstraction in equalities. In particular, since Bedwyr does not use the axiom rule, the original treatment of ∇ is fully satisfying, and there is no need to consider the lifting approach developed in this thesis, or more liberal treatments of generic quantification.

We described how to support generic quantification by relying on higher-order terms and unification. But the latter is in general an undecidable problem. A common approach in logic programming is to restrict unification to higher-order pattern unification, which can solved efficiently and guarantees a most general unifier. In practice, it suffices for most applications. We follow that approach in Bedwyr, adapting the treatment of higher-order pattern unification due to Nadathur and Linnell [NL05].

Tabling

We introduced tabling in Bedwyr to cut-down exponential blowups caused by redundant computations, and to detect loops during proof-search. The first optimization is critical for applications such as weak bisimulation checking. The second one proves useful when exploring reachability in a cyclic graph.

Tabling is currently used in Bedwyr to experiment with proof search for inductive and coinductive specifications: it is the only place where fixed points are not only treated as retracts. A loop over an inductive predicate that would otherwise cause a

divergence can be categorized using tabling as a failure. Similarly, in the co-inductive case, loops yield success. This interpretation of loops as failure or success is not part of μLJ, but such cyclic proofs could be justified by being transformed into μLJ derivations using explicit (co)inductions.

Inductive proof-search with tabling is implemented effectively in provers like XSB [SSW+06] using, for example, suspensions. The implementation of tables in Bedwyr fits simply in the initial design of the prover but is much weaker. We only table a goal in Level-1 when it does not have free occurrences of variables introduced by an existential quantifier; and in Level-0 when it does not have any free variable occurrence. Nevertheless, this implementation of tabling has proved useful in several cases, ranging from graph examples to bisimulation.

It should be noted that tabling involves comparing formulas, which raises the question of the identity of generic judgments. Currently, Bedwyr requires exact matches of the generic contexts, strictly implementing the original minimal generic quantification.

7.1.2 Examples

We give here a brief description of the range of applications of Bedwyr. We refer the reader to `http://slimmer.gforge.inria.fr/bedwyr` and the user manual for Bedwyr [BGM+06] for more details about these and other examples.

Finite failure

Negation of a Level-0 formula P can be written as the Level-1 formula $P \supset \bot$. Operationally, this negation is provable in the Level-1 prover if all attempts to prove P in the Level-0 prover fail. For example, the formula $\forall y.\ ((\lambda x.x) = (\lambda x.y) \supset \bot)$ is a theorem: *i.e.,* the identity abstraction is always different from a constant-valued abstraction. A less trivial example, using the specification of simply typed λ-calculus from Example 6.1, would be that $(\lambda x.xx)$ cannot be typed:

$$\forall t \forall t'.\ \{nil \vdash_{\Lambda^\to} (abs\ t\ (\lambda_{tm} x.\ app\ x\ x)) : t'\} \supset \bot$$

Model-checking

If the two predicates P and Q are defined using Horn clauses, then the Level-1 prover is capable of attempting a proof of $\forall x.\ Px \supset Qx$. This covers most (un)reachability checks common in model-checking. Related examples in the Bedwyr distribution include the verification of a 3 bits addition circuit and graph cyclicity checks. Example 6.1 also falls in that category:

$$\forall \alpha \forall \beta.\ \{nil \vdash_{\Lambda^\to} (abs\ \alpha\ (\lambda_{tm} x.\ abs\ \beta\ (\lambda_{tm} y.\ x))) : arrow\ \alpha\ (arrow\ \beta\ \beta)\} \supset \alpha = \beta$$

Simulation in process calculi

If the Level-0 formula $P \xrightarrow{A} Q$ specifies a one-step transition (process P does an action A and results in process Q), then simulation can be written in Bedwyr as fol-

lows [MMP03]:

$$sim\ P\ Q \triangleq \forall A \forall P'.\ P \xrightarrow{A} P' \supset \exists Q'.\ Q \xrightarrow{A} Q' \wedge sim\ P'\ Q'$$

In dealing with the π-calculus, where bindings can occur within one-step transitions, there are two additional transitions that need to be encoded: $P \xrightarrow{\downarrow X} P'$ and $P \xrightarrow{\uparrow X} P'$, for bound input and bound output transitions on channel X. In both of these cases, P is a process but P' is a name abstraction over a process. The full specification of (late, open) simulation for the π-calculus can be written using the following [MT05]:

$$\begin{aligned} sim\ P\ Q \triangleq\ & [\forall A \forall P'\ .\ P \xrightarrow{A} P' \supset \exists Q'.\ Q \xrightarrow{A} Q' \wedge sim\ P'\ Q'] \wedge \\ & [\forall X \forall P'\ .\ P \xrightarrow{\downarrow X} P' \supset \exists Q'.\ Q \xrightarrow{\downarrow X} Q' \wedge \forall w.sim\ (P'w)\ (Q'w)] \wedge \\ & [\forall X \forall P'\ .\ P \xrightarrow{\uparrow X} P' \supset \exists Q'.\ Q \xrightarrow{\uparrow X} Q' \wedge \nabla w.sim\ (P'w)\ (Q'w)] \end{aligned}$$

Notice that the abstracted continuation resulting from bound input and bound output actions are treated by the $\forall$-quantifier and the ∇ quantifier, respectively. In a similar way, modal logics for the π calculus can be captured [Tiu05]. Interestingly, if the occurrences of *sim* are tabled during proof search, the resulting table contains an actual simulation. Bisimulation is easily captured by simply adding the symmetric clauses for all those used to define *sim*.

7.2 Taci / μLJ

Taci is a framework for interactive proof-search, currently implementing μLJ and some variations of it. It is low-level in that one has to implement in OCaml every detail of a logic, but provides a common interface for tactic-based proof development. Taci has been developed with automated proof-search in mind, a difficult task where the ability to interactively experiment with examples and semi-automated attempts can help a lot. Thus, Taci does not focus only on interactive proof-development but is also designed to fully support automated (backward) proof-search tactics. This essentially means that its notion of tactic supports efficient backtracking and complex control flows, which is implemented using a convenient continuation-passing style.

The logic μLJ has been simply implemented in Taci using the same library as Bedwyr for the representation and unification of higher-order terms[1]. The logic offers a set of primitive tactics corresponding to the rules of sequent calculus. On top of that, we designed the tactic `prove` which implements a focused proof-search. Since that automated tactic is only a strategy for applying sequent calculus rules, the issue of soundness is restricted to these core tactics. In the next section, we give a brief overview of the design of the `prove` tactic. It is an early prototype, which clearly does not compete with state of the art inductive theorem provers like ACL2 or Twelf. Nevertheless, we believe that it is interesting, as it demonstrates that an useful tactic

[1] Hence, some limitations of Bedwyr still hold: for example, there is no way to treat left equalities where existential variables occur. The restriction to higher-order patterns is also problematic in the general context of theorem proving, as evoked in Section 6.3.2 with the example of subject reduction.

can be designed while staying very close to proof-theory. Indeed, only a few standard techniques are applied to derive the `prove` tactic from our μ-focused calculus. Based on that promising first step, we hope that our fundamental study of the structure of (co)inductive proofs eventually brings new advances.

7.2.1 The `prove` tactic

The `prove` tactic implements a simplified version of focused proof-search for μLJ, as described in Figure 4.6. These simplifications allow for a relatively fast depth-first semi-decision procedure. Obviously, the `prove` tactic is very incomplete: it adds essential sources of incompleteness to the more technical ones listed above.

Finding (co)invariants

The biggest difficulty with μLJ is obviously to guess the right (co)invariants. It is a vast question, and several techniques exist for guessing more and more clever invariants. But our goal with the `prove` tactic is to find simple proofs efficiently. It is already a non-trivial task involving many choices. As usual in proof-search, the key to succeeding quickly is to fail quickly. Hence, we shall try only one invariant per possible induction site, and otherwise apply unfolding or freezing. That invariant is obtained from the context:

$$\frac{\Sigma;\ \Gamma, S\vec{t} \vdash G \quad \cdot\ ;\ BS\vec{x} \vdash S\vec{x}}{\Sigma;\ \Gamma, \mu B\vec{t} \vdash G} \qquad \text{with} \quad S := \lambda\vec{x}.\ \forall\Sigma.\ \vec{x} = \vec{t} \supset \bigwedge_{H\in\Gamma} H \supset G$$

With that invariant, the first premise is trivially provable, as it is basically an instance of the identity. The second premise is where proof-search follows. This "trivial induction" is also available to the user in interactive mode, simply by not providing an invariant to the `induction` tactic. This can be compared to proof development in Coq: when one uses the induction tactic in Coq, the invariant is always the current goal; if that is not sufficient, one has to generalize the goal, which corresponds to providing Taci's `induction` tactic with an explicit invariant.

We proceed dually for coinduction:

$$\frac{\Sigma;\ \Gamma \vdash S\vec{t} \quad \cdot\ ;\ S\vec{x} \vdash SS\vec{x}}{\Sigma;\ \Gamma \vdash \nu B\vec{t}} \qquad \text{with} \quad S := \lambda\vec{x}.\ \exists\Sigma.\ \vec{x} = \vec{t} \wedge \bigwedge_{H\in\Gamma} H$$

Such trivial (co)inductions suffice for the examples of Section 3.4. For example, when proving $\forall p.\ sim\ p\ p$, the context does provide a coinvariant: $\lambda p_1 \lambda p_2.\ \exists p.\ p_1 = p \wedge p = p_2$, that is $\lambda p_1 \lambda p_2.\ p_1 = p_2$.

Infinite branches

Contractions, performed in the intuitionistic focusing system (*cf.* Figure 4.6) when choosing a focus on the left hand-side, are a source of infinite search. Rather than bounding the number of nested contractions, we remove them from the focusing rule. This gives a linear aspect to derivations, but it still intuitionistic in that, for example, there is never any splitting of the context.

Fixed points are obviously another source of infinitely deep branches. We avoid it by bounding the number of unfoldings allowed on a branch of the derivation being built. Notice that unfoldings on both sides of the sequent have to be taken into account, but not the induction rule, as it is impossible to have an infinite chain of (co)inductions. We use iterative deepening to obtain faster successes: the bound is iteratively increased, from 0 to its maximum — 3 by default. It does not cost much as proof-search with bound n only takes a negligible time compared to the search with bound $n + 1$. And it brings faster successes since a bound higher than necessary yields a longer search time, as the tactic spends more time on wrong attempts.

The obtained strategy is still fairly simple, and would not give very good results without some refinements. The natural next step is to adapt the cost of unfoldings depending on how likely they seem to lead to a success. In our case, we decided to not decrease the bound on *progressing unfoldings*. Intuitively, we say that an argument of a fixed point is progressing when one learns something by unfolding the fixed point when it has a rigid parameter at that position. In that case the unfolding is said to be progressing. For example, *nat* is progressing in its only argument because if it has a rigid parameter then it can be unfolded to either disappear, obtain the absurdity or obtain *nat* for its subterm. Also, list membership (*mem*) is progressing in its list argument: if the head of the list is defined we can go through it immediately. Although currently it is the user who indicates which are the progressing arguments of a fixed point, we plan to infer it in the future[2]. The optimization based on the notion of progress yields very good results, quickly finding proofs for many interesting properties since they rarely require a high bound on non-progressing unfoldings.

Unfortunately, progressing unfoldings can still go wrong, *i.e.,* yield infinite branches. Consider for example *eq*, the inductive specification of equality on natural numbers: on the left hand-side, one can infinitely unfold *eq* x (s x) where x is an universal variable[3]; similarly on the right hand-side, one can infinitely unfold *eq* X (s X), looking for a successful instantiation of the existential variable X. Such problems rarely occur, and are avoided without impacting on the benefit of progressing unfoldings, by bounding the number of progressing unfoldings on a given formula and its descendants — applying this limitation on a formula-basis rather than globally on the sequent avoids fairness problems. Practical experiments show that this bound can be set quite high (10 in Taci). Indeed, when this kind of situation occurs, it typically quickly consumes the bound without generating side branches.

Organization

We have introduced all the ingredients of the `prove` tactic, but a couple remarks can be made about how they are organized in the focusing framework.

The implementation of the synchronous phase is straightforward, but the same is not true of the asynchronous phase. A novelty of focused proof-search with fixed

[2] In some cases it is good to be able to *not* set progress on a progressing fixed point: for example, when working with the Ackermann function, it avoids that Taci tries to partially unfold its computation.

[3] Although the infinite behavior is obvious in that cyclic example, it is simple to produce up examples which do not cycle. Zach Snow actually observed that the halting problem could be reduced to that of detecting non-terminating progressing unfoldings.

points is that backtracking becomes necessary in the asynchronous phase, because of the choice on fixed points: they might be frozen, unfolded or (co)inducted on. This necessary difficulty should be kept away from the usual asynchronous steps, so that they can still be done without backtracking. Backtracking can also be avoided on progressing unfoldings as it never seems useful to not take these invertible steps[4]. In practice, the following organization of the asynchronous phase yields a reasonably fast search:

1. First perform the asynchronous steps that are not related to fixed points, and progressing fixed point unfoldings. This step is repeated on bodies obtained from unfoldings, and would for example obtain *nat x* from *nat* (*s* (*s x*)). There is no backtracking on that step.

2. Then try for each remaining asynchronous fixed point, in the following order, to either freeze, unfold or (co)induct on it with the trivial invariant. There is some backtracking on that choice. After each attempt, come back to Step 1.

This organization of the asynchronous phase usefully restricts the number of alternatives to backtrack on. However, this comes at a cost. Indeed, since the (co)inductions use the context as invariant, their success depends on when they are performed. Such (co)inductions commute with some asynchronous rules (*e.g.,* $\forall R$, $\exists L$, $\wedge L$) but it is not true, for example, of the $\wedge R$ rule. It is possible that $P \wedge Q$ is an invariant for some operator while P and Q alone are not, for example in inductions using P (resp. Q) at some rank to establish Q (resp. P) at the next one. We accept that problem rather than introducing the cost of more attempts.

At this point, one might wonder what is the benefit of focusing. Although we have a theoretical system that is complete, we had to make several trade-offs and restrictions in order to obtain a practical proof-search strategy. We insist that focusing is still useful as an underlying framework, as it achieves the critical task of reducing the redundancies in proof-search by identifying choices upon which backtracking is useless. This is a huge optimization, especially when dealing with fully synchronous fixed points, which are very common.

From the previous discussion, it should be clear how important it is to precisely control where backtracking occurs or not. It turns out to be difficult to obtain exactly the right behavior using the naive approach consisting in designing elementary blocks, *e.g.,* synchronous or asynchronous steps, and combining them using common combinators like `then`, `cut`, `orelse` and `repeat`. Instead, the correct control flow has to be specially implemented. For that task, the continuation-based tactic system reveals both flexible and efficient.

Lemmas

Sometimes, the simple invariant generation used by `prove` will fail. For example, define *sublist L L′* to hold if L' can be obtained by concatenating elements before and after L, and consider the following theorem: $\forall l.\ list\ l \supset sublist\ l\ l$. It is obvious, because

[4]Progressing unfoldings could exhaust their bound and cause proof-search to fail, but that problem is negligible compared to the state explosion caused by backtracking.

it suffices to concatenate the empty list on both sides of l to obtain itself. However $\lambda l.\ sublist\ l\ l$ is not an invariant of *list*, so Taci will fail to prove this automatically. In order to obtain a derivation of that theorem, one must use as the invariant the fact that the empty list is neutral for concatenations.

An interesting way to overcome that limitation is to introduce support for lemmas. This is done in a very restricted way: at the end of the asynchronous phase, instead of proceeding with the synchronous one, the system attempts to conclude fully from lemmas. This is done by cutting in one or several lemmas and searching for a proof without doing any unfolding at all. Again, the maximal number of lemmas to use shall be limited by a bound to avoid infinite search.

This brings a natural solution to the our example. Typical lemmas about lists would be that the empty list is neutral for concatenation on both sides. Then, if the system attempts to use these lemmas instead of inducting, it will indeed obtain the facts that allow to complete the derivation.

Now, let us ignore for a moment the bounds on unfoldings and the missing contractions in our strategy. Then we have in fact a general recipe: every time an induction fails when it should hold, one can first establish the corresponding lemma, thus informing the system about a new non-trivial invariant. This technique is enough for obtaining any theorem: if a lemma itself cannot be proved automatically, it must be because another non-trivial induction is needed, which can in turn be expressed as a lemma first. The essential difficulty here does not lie in the technical details that we ignored. It is to find where, among lots of failed attempts, the automated strategy was not smart enough, and invent the right lemma to help it.

Generic quantification

In the previous chapter, we refined the proof-theoretical design of minimal generic quantification, showing with the logic $\mu\mathrm{LJ}^{\nabla}$ that minimal generic quantification could be treated in an expressive way. In that system, generic quantification is not treated as a logical connective, and is hence very orthogonal to the rest of the logic. From the implementation viewpoint, it is easy to obtain an implementation of $\mu\mathrm{LJ}^{\nabla}$ from one of μLJ: it basically consists in implementing the transformation ϕ which eliminates toplevel occurrences of the ∇ quantifier. Moreover, the `prove` tactic is then automatically extended into an useful tool when working with generic quantification.

7.2.2 Examples

We show here a few examples of fully automated use of Taci. Of course, an interest of our system is that the user can guide the prover when the automated tactics fail. We invite the reader to visit the project's homepage [BSV08] to learn more about our current set of examples.

- One can describe natural numbers, and relational specifications of several arithmetic operations, using fixed points. In that domain, Taci proves automatically the examples from Section 3.4. Surprisingly, the totality of *half* is also obtained, although none of the two proofs given in the previous chapters (*cf.* Section 3.4

and Example 5.30) fits into the fragment of trivial proofs that the `prove` tactic builds. Instead, the tactic obtains a derivation that uses an inner induction which establishing $half\ x\ h \vdash \exists h'.\ half\ (sx)\ h'$. Several others theorems can be found in Taci's examples, notably many properties of the addition (commutativity, associativity, totality, etc). However, most properties of the multiplication cannot be proved directly, but would require to use lemmas.

- Another set of examples comes with the common specifications of lists and associated operations, such as length, concatenation, reversal. The automated tactic proves several related properties in a few seconds: associativity of concatenation, involutivity of reversal, as well as properties of reversal and concatenation with respect to the length. We also obtained the correction of the insertion sort algorithm.

- The tactic is useful to prove trivial subgoals as well as bureaucratic lemmas when working with higher-order specifications. For example, it suffices to derive several variations of generic strengthening on context lookup:

$$\forall\Gamma\forall x\forall t.\ (\nabla a.\ \langle x, t\rangle \in \Gamma) \supset \langle x, t\rangle \in \Gamma$$

$$\forall\Gamma\forall x\forall t'.\ (\nabla a.\ \langle x, (t'\,a)\rangle \in \Gamma) \supset \exists t.\ t' = (\lambda a.t) \wedge \langle x, t\rangle \in \Gamma$$

$$\forall\Gamma\forall x'\forall t'.\ (\nabla a.\ \langle (x'\,a), (t'\,a)\rangle \in \Gamma) \supset \exists x\exists t.\ x' = (\lambda a.x) \wedge t' = (\lambda a.t) \wedge \langle x, t\rangle \in \Gamma$$

7.3 Conclusion

We have shown how most of the proof-theoretical notions studied in this thesis are put in practice in the tools Bedwyr and Taci. The logic programming language Bedwyr exploits the closed-world assumption that fixed points reflect, as well as generic quantification, offering model-checking abilities directly on higher-order specifications. Taci implements the full logic $\mu\mathrm{LJ}^{\nabla}$ and relies heavily on two developments of this thesis: the μ-focused system for μLJ and the refined treatment of minimal generic quantification in $\mu\mathrm{LJ}^{\nabla}$. It is an useful tool to develop proofs in that logic, as we did for example in Section 6.3.2, and it constitutes a first example of the practical impact of our focusing system.

Both tools share some limitations. First, the treatment of equalities is limited to higher-order patterns and excludes logic variables from left hand-side equalities. A simple enhancement in that respect would be to delay problematic unifications until they become solvable. This would be especially interesting in Taci, where the incompleteness is less tolerable; moreover, its current architecture should allow it easily. But these problems call for a better solution, maybe a new theoretical approach to equality and quantifiers, that would give a full account of the symmetrical roles of universal and existential variables. Finally, we are interested in experimenting with proof-search procedures other than backwards and depth-first, which would not be limited to a very local treatment of goals; the ideas outlined in Chapter 5 go in that direction.

Not the end.

List of Figures

Index

Bibliography

[AFFM06] Sandra Alves, Maribel Fernández, Mário Florido, and Ian Mackie. The power of linear functions. In *CSL*, pages 119–134, 2006.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

[AP91] J. M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9(3-4):445–473, 1991.

[AvE82] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *J. of the ACM*, 29(3):841–862, 1982.

[Bae05] David Baelde. Logique linéaire et algèbre de processus. Technical report, INRIA Futurs, LIX and ENS, 2005.

[Bae08a] David Baelde. On the expressivity of minimal generic quantification. In A. Abel and C. Urban, editors, *LFMTP 2008: International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, 2008. To appear.

[Bae08b] David Baelde. On the expressivity of minimal generic quantification: Extended version. Technical report, 2008. Available from http://hal.inria.fr/inria-00284186.

[BGM+06] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. *A User Guide to Bedwyr*, November 2006.

[BGM+07a] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. Bedwyr. 2007.

[BGM+07b] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The Bedwyr system for model checking over syntactic expressions. In Frank Pfenning, editor, *21th Conference on Automated Deduction (CADE)*, number 4603 in LNAI, pages 391–397. Springer, 2007.

[BM07] David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In N. Dershowitz and A. Voronkov, editors, *International Conference on Logic for Programming and Automated Reasoning (LPAR)*, volume 4790 of *LNCS*, pages 92–106, 2007.

[Bro06] James Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions*. PhD thesis, University of Edinburgh, November 2006.

[BS07] James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *LICS*, pages 51–62, 2007.

[BSV08] David Baelde, Zach Snow, and Alexandre Viel. Taci: an interactive theorem proving framework. Active development of prototype, 2008.

[Bur86] Albert Burroni. Récursivité graphique (1ère partie) : Catégorie des fonctions récursives primitives formelles. In *Cahiers de topologie et géométrie différentielle catégoriques, XXVII, 1*, 1986.

[Cha06] Kaustuv Chaudhuri. *The Focused Inverse Method for Linear Logic*. PhD thesis, Carnegie Mellon University, December 2006. Technical report CMU-CS-06-162.

[CP02] Iliano Cervesato and Frank Pfenning. A Linear Logical Framework. *Information & Computation*, 179(1):19–75, November 2002.

[CS02] J. Robin B. Cockett and Luigi Santocanale. Induction, coinduction, and adjoints. *Electr. Notes Theor. Comput. Sci.*, 69, 2002.

[DJS93] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Kurt Gödel Colloquium*, volume 713 of *LNCS*, pages 159–171. Springer, 1993.

[DJS95] V. Danos, J.-B. Joinet, and H. Schellinx. LKT and LKQ: sequent calculi for second order logic based upon dual linear decompositions of classical implication. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, number 222 in London Mathematical Society Lecture Note Series, pages 211–224. Cambridge University Press, 1995.

[EP91] Conal Elliott and Frank Pfenning. A semi-functional implementation of a higher-order logic programming language. In Peter Lee, editor, *Topics in Advanced Language Implementation*, pages 289–325. MIT Press, 1991.

[Gac08] Andrew Gacek. The Abella interactive theorem prover (system description). Available from http://arxiv.org/abs/0803.2305. To appear in IJCAR'08, 2008.

[Gen69] Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

[Gim96] Eduardo Giménez. *Un Calcul de Constructions Infinies et son Application a la Verification des Systemes Communicants*. PhD thesis PhD 96-11, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, December 1996.

[Gim98] Eduardo Giménez. Structural recursive definitions in type theory. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings 25th Int. Coll. on Automata, Languages and Programming, ICALP'98, Aalborg, Denmark, 13–17 July 1998*, volume 1443 of *LNCS*, pages 397–408. Springer-Verlag, Berlin, 1998.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir91a] Jean-Yves Girard. A new constructive logic: classical logic. *Math. Structures in Comp. Science*, 1:255–296, 1991.

[Gir91b] Jean-Yves Girard. On the unity of logic. Technical Report 26, Université Paris VII, June 1991.

[Gir92] Jean-Yves Girard. A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu, February 1992.

[Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143, 1998.

[Gir01] Jean-Yves Girard. Locus solum. *Mathematical Structures in Computer Science*, 11(3):301–506, June 2001.

[GMN08] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In F. Pfenning, editor, *23th Symp. on Logic in Computer Science*. IEEE Computer Society Press, 2008. To appear.

[Hen93] H. Hendriks. *Studied Flexibility: Categories and Types in Syntax and Semantics*. PhD thesis, ILLC, Amsterdam, 1993.

[HL78] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[Hod94] Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, May 1994.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.

[Lau02] Olivier Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.

[Lau04] Olivier Laurent. A proof of the focalization property of linear logic. Unpublished note, May 2004.

[LM07a] Chuck Liang and Dale Miller. Focusing and polarization in intuitionistic logic. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 451–465. Springer, 2007.

[LM07b] Chuck Liang and Dale Miller. On focusing and polarities in linear logic and intuitionistic logic. Available via http://hal.inria.fr/inria-00167231/, September 2007.

[LQdF05] Olivier Laurent, Myriam Quatrini, and Lorenzo Tortora de Falco. Polarized and focalized linear and classical proofs. *Ann. Pure Appl. Logic*, 134(2-3):217–264, 2005.

[LR03] Olivier Laurent and Laurent Regnier. About translations of classical logic into polarized linear logic. In *18th Symp. on Logic in Computer Science*, pages 11–20. IEEE Computer Society Press, June 2003.

[Mat98] Ralph Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. PhD thesis, Ludwig-Maximilians Universität, May 1998.

[Mat99] Ralph Matthes. Monotone fixed-point types and strong normalization. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, *Proceedings 12th Int. Workshop on Computer Science Logic, CSL'98, Brno, Czech Republic, 24–28 Aug 1998*, volume 1584, pages 298–312. Springer-Verlag, Berlin, 1999.

[Men87] Paul Francis Mendler. *Inductive Definition in Type Theory*. PhD thesis, Cornell University, 1987.

[Mil92] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14(4):321–358, 1992.

[Mil93] Dale Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *3rd Workshop on Extensions to Logic Programming*, number 660 in LNCS, pages 242–265, Bologna, Italy, 1993. Springer-Verlag.

[Mil96] Dale Miller. Forum: A multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, September 1996.

[Mil00] Dale Miller. Abstract syntax for variable binders: An overview. In John Lloyd and et. al., editors, *Computational Logic - CL 2000*, number 1861 in LNAI, pages 239–253. Springer, 2000.

[MM00] Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.

[MM02] Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. on Computational Logic*, 3(1):80–136, 2002.

[MMP03] Raymond McDowell, Dale Miller, and Catuscia Palamidessi. Encoding transition systems in sequent calculus. *Theoretical Computer Science*, 294(3):411–437, 2003.

[MN87] Dale Miller and Gopalan Nadathur. A logic programming approach to manipulating formulas and programs. In Seif Haridi, editor, *IEEE Symposium on Logic Programming*, pages 379–388, San Francisco, September 1987.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[MS06] Dale Miller and Alexis Saurin. A game semantics for proof search: Preliminary results. In *Proceedings of the Mathematical Foundations of Programming Semantics (MFPS05)*, number 155 in Electr. Notes Theor. Comput. Sci, pages 543–563, 2006.

[MS07] Dale Miller and Alexis Saurin. From proofs to focused proofs: a modular proof of focalization in linear logic. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 405–419. Springer, 2007.

[MT03a] Dale Miller and Alwen Tiu. A proof theory for generic judgments: An extended abstract. In *18th Symp. on Logic in Computer Science*, pages 118–127. IEEE, June 2003.

[MT03b] Alberto Momigliano and Alwen Tiu. Induction and co-induction in sequent calculus. In Mario Coppo Stefano Berardi and Ferruccio Damiani, editors, *Post-proceedings of TYPES 2003*, number 3085 in LNCS, pages 293–308, January 2003.

[MT05] Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, October 2005.

[NL05] Gopalan Nadathur and Natalie Linnell. Practical higher-order pattern unification with on-the-fly raising. In *ICLP 2005: 21st International Logic Programming Conference*, volume 3668 of *LNCS*, pages 371–386, Sitges, Spain, October 2005. Springer.

[PE88] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, June 1988.

[Pie05] Brigitte Pientka. Tabling for higher-order logic programming. In *20th International Conference on Automated Deduction, Talinn, Estonia*, pages 54–69. Springer-Verlag, 2005.

[Pie08] Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th Annual ACM Symposium on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM, 2008.

[Pit03] Andrew M. Pitts. Nominal logic, A first order theory of names and binding. *Information and Computation*, 186(2):165–193, 2003.

[PM96] Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation à diriger les recherches, Université Claude Bernard Lyon I, December 1996.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — A meta-logical framework for deductive systems. In H. Ganzinger, editor, *16th Conference on Automated Deduction (CADE)*, number 1632 in LNAI, pages 202–206, Trento, 1999. Springer.

[PS08] Adam Poswolsky and Carsten Schürmann. Delphin – a functional programming language for deductive systems. In A. Abel and C. Urban, editors, *LFMTP 2008: International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, 2008. To appear.

[San02] Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS02)*, number 2303 in LNCS, pages 357–371. Springer-Verlag, January 2002.

[Sau08] Alexis Saurin. *Une étude logique du contrôle*. Thèse de doctorat, École Polytechnique, 2008.

[SBM07] Zach Snow, David Baelde, and Dale Miller. Taci: an interactive theorem proving framework. 2007.

[SH93] Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.

[SP98] Carsten Schürmann and Frank Pfenning. Automated theorem proving in a simple meta-logic for LF. In Claude Kirchner and Hélène Kirchner, editors, *15th Conference on Automated Deduction (CADE)*, volume 1421 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 1998.

[SSW+06] Konstantinos Sagonas, Terrance Swift, David S. Warren, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Steve Dawson, and Michael Kifer. *The XSB Version 3.0 Volume 1: Programmer's Manual*, 2006.

[Ter04] Kazushige Terui. Light affine set theory: A naive set theory of polynomial time. *Studia Logica*, 77(1):9–40, 2004.

[Tiu04] Alwen Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Pennsylvania State University, May 2004.

[Tiu05] Alwen Tiu. Model checking for π-calculus using proof search. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 36–50. Springer, 2005.

[Tiu06] Alwen Tiu. A logic for reasoning about generic judgments. In A. Momigliano and B. Pientka, editors, *Int. Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'06)*, 2006.

[TNM05] Alwen Tiu, Gopalan Nadathur, and Dale Miller. Mixing finite success and finite failure in an automated prover. In *Proceedings of ESHOL'05: Empirically Successful Automated Reasoning in Higher-Order Logics*, pages 79–98, December 2005.

[UT05] Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In R. Nieuwenhuis, editor, *20th Conference on Automated Deduction (CADE)*, volume 3632 of *LNCS*, pages 38–53. Springer, 2005.