An overview of focusing for least and greatest fixed points in intuitionistic logic

David Baelde

University of Minnesota

Abstract. Least and greatest fixed points, corresponding to inductive and coinductive definitions, are known to be difficult concepts in proof-theory. Recent advances, however, have brought some structure to these problems: The notion of focusing has been extended to support least and greatest fixed points, opening promising avenues. This has been done first in linear logic (µMALL and µLL), but also applies to the intuitionistic setting (µLJ). In this paper, we fully review the current state of focusing in µLJ, its successes in automated reasoning as well as its puzzles, on which we provide new insights by studying how our focusing mechanisms apply to related systems.

1 Introduction

Inductive and coinductive specifications are pervasive in computer science, and their importance is still growing, in particular the use of coinductive techniques to analyze the behavior of processes. We are interested in developing a formal support for reasoning about such specifications, which involves (co)induction. Proof-theory is an appealing framework for such a task. Despite its obvious use for reasoning about specifications, it can support prototyping by computing on specifications, using the logic programming paradigm. Extending this paradigm a little bit, it is also possible to carry out some verification or model-checking tasks directly in proof-theory. Moreover, proof theory provides an uniform and modular understanding of various aspects of several logics, notably through their sequent calculus presentations.

The proof theoretical treatment of fixed points contains some difficulties. Notably, the (co)induction principles involve (co)invariants that might have little to do with the formulas to be proved. This generalization breaks the subformula property and makes proof-search significantly more difficult. In that context, it is interesting to study focusing, a proof-theoretical technique that reveals important structures in proofs, notably giving a better understanding of the choices involved in proof-search. Focusing is known to enable logic programming interpretations of various logics. Going further, we have established its usefulness in various proof-search-based tasks involving (co)inductive specifications. Focusing does not suppress the inevitable difficulties caused by fixed points, but organizes them in ways that enable practical applications.

The study of focusing is strongly tied to linear logic. Focusing fixed points has first been studied in the purely linear setting of µMALL [BM07] and the extension of those observations to intuitionistic logic [Bae08] is less known. In this paper, we review those results without any reference to linear logic, to make it more accessible and perhaps spread focusing to other communities.
We present the logic \( \mu \text{LJ} \) in Section 2 and show in Section 3 how focused proof systems can be designed for that logic, and how they can be exploited. In Section 4 we make new observations in an attempt to shed some light on old puzzles.

2 The logic \( \mu \text{LJ} \)

The logic \( \mu \text{LJ} \) is the extension of first-order intuitionistic logic with inductive and coinductive definitions given using least and greatest fixed points. The proof system for \( \mu \text{LJ} \) contains familiar rules for inductive and coinductive inference based on the selection of invariants. It notably provides the intuitionistic version of Peano’s arithmetic. The logic \( \mu \text{LJ} \) results from a line of work on definitions \([\text{Gir}92,\text{SH}93]\) and (co)induction \([\text{MM}00,\text{MT}03]\). Instead of definitions, the presentation using \( \mu \) and \( \nu \) used in this paper makes for a more direct proof theoretical study and naturally supports mutually (co)inductive definitions.

We consider the following simply typed language of formulas:

\[
P ::= P \land P | P \lor P | P \supset P | \bot | \top |
| \exists x. P | \forall x. P | s \stackrel{\equiv}{=} t | \mu_{\gamma_1, \ldots, \gamma_n} (\lambda p. \lambda x. P)t | \nu_{\gamma_1, \ldots, \gamma_n} (\lambda p. \lambda x. P)t.
\]

The syntactic variable \( \gamma \) represents a term type, e.g., natural numbers or lists. The quantifiers have type \((\gamma \to o) \to o\) and the equality has type \(\gamma \to \gamma \to o\). The least fixed point connective \( \mu \) and the greatest fixed point connective \( \nu \) have type \((\tau \to \tau) \to \tau\) where \( \tau \) is \(\gamma_1 \to \cdots \to \gamma_n \to o\) for some arity \(n \geq 0\). We shall always elide the references to \( \gamma \), assuming that they can be determined from the context when it is important to know their value. Note that we do not consider atoms, i.e., predicate constants: although \( \mu \text{LJ} \) accommodates them without any problem, atoms are often unnecessary since fixed points play their role in practice, and thus we leave them out for simplicity.

Formulas with top-level connective \( \mu \) or \( \nu \) are called fixed point expressions. Fixed points can be arbitrarily nested and interleaved — that is, we can have mutually recursive definitions. The first argument of a fixed point connective is a predicate operator expression, called its body, and shall be denoted by \( B \). In order for the logic to enjoy consistency and other useful properties, all fixed point bodies are required to be monotonic, i.e., there should be no negative occurrence of the bound predicate variable \( p \) in \( \lambda p. \lambda x. Bp \).

Example 1. Assuming a term type \( n \) and two constants \( 0 : n \) and \( s : n \to n \), the natural number predicate \( \text{nat} \) of type \( n \to o \) can be defined as the inductive expression \( \mu B_{\text{nat}} \), where \( B_{\text{nat}} \) is defined as \( \lambda N. \lambda x. x = 0 \lor \exists y. x = s \cdot y \land N \cdot y \).

The inference rules of \( \mu \text{LJ} \) deal with usual first-order intuitionistic sequents, of the form \( \Sigma ; \Gamma \vdash P \) where \( \Sigma \) is a set of universal (eigen)variables \( x_1, \ldots, x_n \) and \( \Gamma \) is a set of formulas \( P_1, \ldots, P_m \) (i.e., contraction is implicit). The logical reading of such a sequent is \( \forall x_1 \ldots \forall x_n. (P_1 \land \cdots \land P_m \supset P) \). The full system is presented in Figure 1, and we detail below its design.
Thus, the least prefixed point is a fixed point, an approach to negation-as-failure. More generally, equality provides our proof system with a treatment of equality that is stronger than Leibniz equality, as it notably expresses the injectivity of term constructors. The least fixed point \( \mu \) is characterized as the least of the prefixed points of \( \mu \). This means that unfolding (on either side) is invertible. In particular, instances of the initial rule on fixed points can always be expanded by unfolding. It is important to note, however, that they cannot in general be totally removed, as repeated expansions might not terminate. Finally, all those observations about least fixed points can be carried to greatest fixed points which are defined in a very symmetric way as the greatest (right-rule) of the post-fixed points (left rule).

The inference rules of \( \mu LJ \) are the usual ones for the propositional connectives and first-order quantifiers. The left and right introduction rules for equality date back to [Gir92,SH93]. In the left equality rule, \( \text{csu} \) stands for complete set of unifiers. This set can be restricted to have at most one element when terms are first-order but might be infinite if terms are interpreted modulo some algebraic theory or if they are simply typed \( \lambda \)-terms. The application of a substitution to the signature of a sequent consists in removing instantiated variables and adding newly introduced ones; the application to the rest of the sequent simply propagates it to the terms of every formula. Note that this treatment of equality is stronger than Leibniz equality, as it notably expresses the injectivity of term constructors. More generally, equality provides our proof system with an approach to negation-as-failure: if the equality \( t = t' \) is a failure (that is, \( \text{csu}(t,t') \) is empty) then the equality left rule yields a successful proof (that is, there are no premises to the rule).

The least fixed point \( \mu B \) is characterized as the least of the prefixed points of \( B \). The right rule, called unfolding, expresses \( B(\mu B)t \supset \mu Bt \), and the left induction rule expresses that \( \mu B \) entails any prefixed point \( S \), also called an invariant. Notice that the universal variables \( x \) in the induction rule are new. From the induction rule one can always derive a left unfolding rule for \( \mu \), using the invariant \( B(\mu B) \):

\[
\Sigma; \Gamma, B(\mu B)t \vdash P \\
\Sigma; \Gamma, \mu Bt \vdash P
\]

Thus, the least prefixed point is a fixed point, \( i.e. \), \( \mu Bx \) and \( B(\mu B)x \) are provably equivalent. This means that unfolding (on either side) is invertible. In particular, instances of the initial rule on fixed points can always be expanded by unfolding. It is important to note, however, that they cannot in general be totally removed, as repeated expansions might not terminate. Finally, all those observations about least fixed points can be carried to greatest fixed points which are defined in a very symmetric way as the greatest (right-rule) of the post-fixed points (left rule).
Example 2. In the particular case of \textit{nat}, the induction rule with invariant $S$ yields the usual induction principle:

\[
\Sigma; \Gamma, \text{nat } t \vdash P \\
\frac{\vdash S(0) \quad y; S(y) \vdash S(s\ y)}{\exists L, \forall L, \land L, \land L, =L \\
\Sigma; \Gamma, S(0) \vdash P \\
\frac{\vdash S(y) \quad y; S(y) \vdash S(s\ y)}{\exists L, \forall L, \land L, \land L, =L \\
\Sigma; \Gamma, S(s\ y) \vdash P}
\]

Computationally, (co)induction corresponds to iteration, and can for example be related to a restricted form of (co)\texttt{fix} constructions in Coq. The cut reductions are described in [Bae08], but there is currently no complete proof of cut-elimination for first-order $\mu$LJ. However, the technique used for $\mu$MALL [Bae09] should be adapted easily. Also, Clairambault [Cla09] gave a proof of cut-elimination for the propositional fragment using game semantics, and Momigliano & Tiu [MT03] treated a similar system with stratified definitions.

For brevity, we shall omit the signature $\Sigma$ from the sequents in the next sections; its treatment should be clear from these paragraphs.

3 Focusing

Sequent calculus is an elegant framework for presenting a logic, exhibiting the symmetries of its connectives. However, raw sequent calculus proofs are known to contain irrelevant details that obfuscate their study and make proof-search impractical: in a given proof, the relative order of many rule applications does not matter, as those rules can often be permuted to obtain a (syntactically) different proof. Further, it is sometimes useful to ignore some details of a proof, such as sequences of rules which can be easily recovered, for example to reduce its size — a particular instance of this is the attempt to separate computation and deduction, for example in deduction modulo.

Some observations allow to restrict the amount of noise, such as the well-known notion of invertibility or the more recent notion of backchaining in uniform proofs. Focusing generalizes those ideas into a single elegant framework. Focused proofs rely on a classification of connectives (and atoms) into asynchronous and synchronous ones, and proceed in an alternation of two phases: in the asynchronous phase, all asynchronous connectives are introduced, in an order-irrelevant way; in the synchronous phase, a formula whose toplevel connective is synchronous is selected, and its outermost layer of synchronous connectives is fully introduced, which usually involves important choices.

Focusing has first been developed for linear logic [And92], but has been shown to also apply to intuitionistic and classical logics as well as other extensions. In linear logic, each connective can only be assigned one polarity (asynchronous or synchronous) but a given atom can be assigned any polarity since no logical structure forces a choice (this flexibility can be exploited to obtain backward or forward chaining in focused proof-search [LM09]). In intuitionistic logic, and even more in classical logic, some connectives can have both a synchronous and an asynchronous version, and it is possible to use them both in the same proof, which can be useful as a way for the specifier to control proof-search. This is just the next level of the well-known fact that equivalent statements can lead to different proofs and proof-search behavior (consider for example, when $y$ and $z$ are natural numbers, $\forall x. x = y + z \supset Px$ and $\exists x. x = y + z \land Px$). In this
short paper, we concentrate on fixed points, and do not present a system with maximum flexibility, but insist that our mechanisms can be combined very easily with those.

It is often thought that asynchrony is the same as invertibility, synchrony being non-invertibility. There is indeed a coincidence in many logics, but it should not be taken as the rule. In any case, we shall see that this viewpoint cannot be satisfied when dealing with fixed points. A better perspective in our opinion lies in the common interpretation of logic as a game between a prover and a refuter. Synchronous connectives are those which are played by the prover, asynchronous ones are played by the refuter. Completeness of focusing expresses that one player's consecutive choices can be grouped without loosing strategies. A common reformulation of this game is to consider a program playing against its environment. In that viewpoint, an interesting particular case in this viewpoint is that of data, made of pairs (conjunction) and variant types (disjunction). Such objects are built by the program, without any interaction with the environment: since there is no way of telling when the environment might inspect a pair, there is no point delaying the construction of its components — unless it is a lazy pair of type \((\top \to A) \land (\top \to B)\) which is not pure data anymore. Hence, building a data item must be done fully in a single synchronous phase. That functional programming view is backed by an important fact about focusing: the process of transforming a proof into a focused one does not change its essence, that is its computational behavior in cut reductions. This is known for linear, intuitionistic and classical logics (for which there are in fact focusing proofs based directly on cut-elimination) but remains an important conjecture for fixed points.

Having given some background on focusing, we now present focused systems for \(\mu LJ\). The first system, called \(\mu\)-focused, treats least fixed points as synchronous, consistently with linear logic observations. But we also present a \(\nu\)-focused system, which is more surprising but nevertheless complete.

### 3.1 The \(\mu\)-focused system

We present in this section a focused proof system for \(\mu LJ\) that treats least fixed points as synchronous. This corresponds to the intuition that data types (such as natural numbers, \(\mu N. \lambda x. x = 0 \lor \exists y. x = s y \land N y\) or simply \(\mu N. \top \lor N\)) should be purely synchronous. It also corresponds to observations in linear logic (where synchronous connectives usually preserve positive formulas, \(\mu\), and to the treatment of fixed points in parity games, as we shall see in Section 4.1. A key observation here is that although all fixed point rules are invertible (by choosing unfolding invariants in the case of (co)induction) we cannot obtain a complete proof system by eagerly applying those rules. Instead, a freezing mechanism has to be introduced, reflecting the fact that at some point, one has to stop exploring the structure of fixed points and treat them like atoms, using the initial rule.

**Definition 1 (Polarities for \(\mu LJ\)).** The connectives \(\land, \lor, \exists, =, \mu\) are synchronous while the connectives \(\forall, \supset, \nu\) are asynchronous. A synchronous (resp. asynchronous) formula is one whose top-level connective is synchronous (resp. asynchronous). If every connective of a formula is synchronous (resp. asynchronous), it is called fully synchronous (resp. asynchronous). Finally, a fixed point formula can be annotated as
frozen, which is denoted by \((\mu Bt)^*\) and \((\nu Bt)^*\), in which case it is neither synchronous nor asynchronous.

Figures 2, 3 and 4 present µLJF, a focused proof system for µLJ. There are two kinds of sequents: the unfocused sequent is written \(\Gamma \vdash P\) (as before) and the focused sequent is written with a distinguished formula (the focus) as either \(\Gamma \vdash \downarrow P\) or \(\Gamma \vdash \uparrow P\). In each of these sequents, \(\Gamma\) is a multiset of formulas. There is an unsurprising symmetry between left and right hand-sides of sequents: a synchronous connective is treated as asynchronous on the left and vice-versa. The asynchronous phase contains sequents of the form \(\Gamma \vdash P\) and introduces asynchronous connectives on the right and synchronous ones on the left. The synchronous phase deals with sequents containing one distinguished formula that is under focus. When the focus is on the right (\(\Gamma \vdash \downarrow P\)) only the synchronous connectives of \(P\) can be introduced. When the focus is on the left (\(\Gamma \vdash \uparrow P\)) only asynchronous connectives of \(P\) can be introduced. The articulation between the two phases is allowed only when no other rule applies: the asynchronous phase ends when no synchronous formula remains on the left, and the conclusion is synchronous; the synchronous phase ends when the focus is on the left on a synchronous formula, or on the right on an asynchronous one. Finally, the structural rule of contraction is used (implicitly) only in the rule establishing a left focus formula — and thus, only for asynchronous formulas.

Each fixed point has two rules per phase: one of these rules treats the fixed point as a structured formula; the other treats it as an atom. The synchronous rules are unfoldings and initial rules and the asynchronous rules are (co)induction and freezing. A strong constraint of the asynchronous phase is that it requires that any least fixed point hypothesis (and greatest fixed point conclusion) is either immediately used for (co)induction (which includes unfolding) or frozen, in which case it can never again be unfolded or used for induction: it can only be used in an initial rule later in the proof. Also note that when one focuses on a fully synchronous least fixed point, such as \(nat\), \(list\) or any encoding of a Prolog computation, focus can never be released. Hence, the proof has to be completed in that phase, eventually reaching units, equality, or the initial rule if an appropriate frozen side-formula is available.

Theorem 1 (Soundness and completeness). A sequent \(\Gamma \vdash P\) is provable in µLJ if and only if it is provable in the µ-focused system for µLJ.
Proof. Soundness is trivial: erasing the focusing annotations from a focused proof, one obtains a standard proof. We prove completeness using the technique of the focalization graph [MS07]: we place ourselves in an unfocused logic and provide permutations that will be used to transform any proof in that system into a proof whose rules are applied in a focused way, allowing the addition of focusing annotations.

Our unfocused logic must however be a more structured version of $\mu$LJ. This intermediate, dyadic proof system is equipped with a freezing rule, requires a frozen fixed point in its initial rule, and forbids any other logical rule on frozen fixed points. Adapting proofs to comply to these changes is easy. More importantly, our system must give a precise account of contractions, by separating the left hand-side of sequents into a linear and a non-linear zone, in the style of Andreoli’s dyadic sequents for linear logic: the non-linear zone is only used as a storage from which formulas are copied (contracted) to the linear zone where logical rules are applied. That system must guarantee that synchronous hypothesis are never moved to the non-linear zone (thus, they are never contracted) and that asynchronous hypothesis are always moved to the non-linear zone. Due do the lack of space, we do not detail the rules of that intermediate proof system nor the transformation of $\mu$LJ proofs into it.

The heart of the focusing technique can then be applied. It relies on two groups of rule permutations. First, synchronous rules are required to permute with each other. This is easy to check: the only non-standard case is that of least fixed point unfolding, which permutes very easily with other rules. Second, we need to be able to permute asynchronous rules under any other. Here, a problematic case is the permutation of asynchronous fixed point rules under a left disjunction rule\(^1\). Consider a derivation starting as follows:

\[
\frac{\Gamma, P, \mu Bt \vdash Q}{\Gamma, P \vee P', \mu Bt \vdash Q}
\]

\(^1\) Precisely, the problem comes from additive branching, which would also be found with an asynchronous version of conjunction, and treated in the same way for focusing.
We need to be able to transform it so that it starts with a fixed point rule: either freezing or induction — in the case of a greatest fixed point occurring on the right hand-side this would be freezing or coinduction. The problem is that different rules could be applied on the two sides: freezing and induction, or two inductions with different invariants. To avoid the first case, we can postpone freezing by unfolding. To deal with the second case, we observe that if \( S \) and \( S' \) are invariants of (resp. coinvariants) of \( B \) then so is \( S \land S' \) (resp. \( S \lor S' \)). These two transformations can be iterated as a pre-processing phase, which we call balancing, after which it will be guaranteed that a given asynchronous fixed point occurrence is used in exactly one way throughout the proof, which enables the permutations. We refer the reader to [Bae08] for a full presentation.

Although not visible in the statement of completeness, the asynchronous rules can be applied in any order — permuting them actually leaves the proof essentially unchanged. As usual, the completeness of focused proofs justifies a reading of logic based on synthetic connectives and synthetic introduction rules. A synthetic introduction rule for a synthetic synchronous connective is a big-step rule that has a focused sequent as its conclusion and which extends upwards until there are only unfocused sequents present. Dually, a synthetic introduction rule for a synthetic asynchronous connective is a big-step rule that has an unfocused sequent as its conclusion and which extends upwards until no asynchronous rule can be applied. Note that this is especially powerful with fixed points, since we can now have synthetic introduction rules built from unbounded numbers of micro-rules. An interesting particular case of this is that of fully synchronous formulas, such as \( \text{nat} \) and more generally any Prolog-style computation, which constitute a synthetic unit, with an infinity of synthetic introduction rules.

3.2 The \( \nu \)-focused system

It is possible to design another complete focused proof system, where (co)induction is synchronous, following the intuition that it’s an important choice. However, freezing is still needed, so the asynchronous phase still contains choices. This system would only differ from the previous one in its fixed point rules:

\[
\begin{align*}
\Gamma, B(\nu B)t & \vdash P & & \Gamma, (\nu B)t^* & \vdash P & & \Gamma \vdash B(\mu B)t & & \Gamma \vdash (\mu B)t^* \\
\Gamma \triangledown S t & \vdash P & & \Gamma \triangledown B S x & \vdash S x & & \Gamma \triangledown \mu B t & \vdash P & & \Gamma \triangledown (\mu B)t^* & \vdash P & & \Gamma \triangledown v B t & \vdash (\nu B)t^* & \vdash \nu v B t
\end{align*}
\]

The proof of completeness follows the same general structure as before, with an important conceptual difference: in \( \mu \)-focusing contraction is admissible for asynchronous formulas (synchronous hypothesis) in the strong sense that \( P \) linearly implies \( P \land P \). This is not the case anymore, which complicates the translation to the dyadic system where synchronous hypothesis cannot be treated non-linearly. The solution is to perform this translation at the same time as the balancing process: once a greatest fixed point hypothesis is balanced, its use (freezing or unfolding) can be anticipated before any contraction, thereby pushing contraction to subformulas. Apart from that subtlety, balancing is in itself simpler than in \( \mu \)-focusing because we only balance the number of unfoldings, and do not have to merge (co)invariants.
Finally, the previous observations can be combined to yield a mixed $\mu$ and $\nu$-focusing system, where explicit polarity annotations on fixed point connectives indicate to treat some least (resp. greatest) fixed points as synchronous and others as asynchronous.

3.3 Applications

The structures provided by focused proofs for $\mu$LJ have been used in two automated reasoning tools.

Bedwyr [BGM+07] is a logic programming language based on focused proof-search for a fragment of $\mu$LJ. It allows implications with purely synchronous hypothesis, and performs exhaustive case analysis on those. This model-checking behavior allows to explore the possible outcomes of any Prolog-like computation, but can also be used in richer ways, e.g., to check bisimilarity. Bedwyr searches for “finite behavior” proofs, that do not contain any instance of the initial rule or (co)induction, but only proceed by unfolding fixed points: this corresponds to a focused proof-search with both $\mu$ and $\nu$ being synchronous and without a freezing rule. Although in that simple setting there is no difference between least and greatest fixed points, Bedwyr also features a tabling extension that allows it to recognize loops in proof-search and classify them as successes for greatest fixed points or failures for least fixed points.

The second tool is Tac [BMS10], a proof assistant for the full logic $\mu$LJ (extended with the $\nabla$ quantifier to support reasoning over higher-order abstract syntax specifications). It was developed with the main purpose of studying automation in that context, and features a powerful automatic tactic that performs $\mu$-focused proof-search using (co)induction. Although focusing plays the usual role of organizing choices in proof-search, it is also useful here to help telling apart computation and deduction, which is critical to powerful proof-search in such a rich setting: complex deduction steps should be limited, but relatively long computations need to be carried out even in simple proofs.

4 Beyond $\mu$LJ

Having presented and discussed focusing mechanisms for $\mu$LJ, we now turn our attention to variant systems, in order to test the robustness of our observations.

4.1 Games and infinitary proofs

Let us consider a radically different framework for reasoning with least and greatest fixed points: infinite proofs. The system, related to [San02,Bro05], consists of taking as only fixed point rules the unfoldings on both sides, and considering not only finite proofs but also infinite ones, i.e., trees with infinite branches. There is no more initial rule: identities are instead infinitely expanded. At this point, we have erased the difference between least and greatest fixed point and obtained an inconsistent logic. Consider
for example the following infinite derivation:

\[
\vdash (\mu P. P) \quad \vdash (\mu P. P) + \bot \\
\vdash (\mu P. P) \quad \vdash (\mu P. P) + \bot \\
\vdash \bot
\]

The last ingredient is a gain condition, which requires that all infinite paths in a derivation go through infinitely many unfoldings of the same\(^2\) greatest fixed point on the right hand-side, or least fixed point on the left hand-side. Only the right subderivation of the above example satisfies this condition. The gain condition restores cut-elimination, by making it impossible that the cut-formula is the only active formula in both sub-derivations, thereby allowing the reduction of any cut until at least one rule is permuted under it, defining a productive process that yields a well-defined cut-elimination procedure for infinite derivations.

The resulting system might not qualify as a satisfying proof system: If proofs cannot even be represented finitely, how to check them mechanically? Note also that infinite proofs invalidate Gödel’s incompleteness argument. Such systems are in fact more related to semantics than to proofs. Indeed, the gain condition corresponds to the one found in parity games [San02, Cla09].

Let us now consider how to focus such derivations. The freezing rule would disappear, because there is no more initial rule, and, more fundamentally, no necessity to avoid infinite (invertible) expansions of fixed points. Rule permutations regarding fixed points have become totally trivial: even more than with \(\mu\) LJ, rule permutabilities are not going to force polarities. However, extending the focalization graph technique is not obvious at all, since it relies on an induction on the synchronous trunk of a derivation, i.e., its final section consisting only of synchronous rules. The key observation here is that the gain condition forces that trunk to be finite if and only if least fixed points are considered synchronous. Hence, the same focalization technique can be applied to obtain a focused system without freezing, with \(\mu\) treated as synchronous and \(\nu\) as asynchronous. But there is no more flexibility in the polarity of fixed points in that setting.

### 4.2 Variations on the (co)induction rule

We finally turn our attention to other formulations of the (co)induction rules. The rules presented above might seem restrictive, and one could instead consider the following generalized rules:

\[
\begin{align*}
\Sigma; \Gamma, S t & \vdash P \\
\Sigma; x; \Gamma, BS x & \vdash S x \\
\Sigma; \Gamma, \mu B t & \vdash P \\
\Sigma; \Gamma, \nu B t & \vdash S x \\
\end{align*}
\]

\(^2\) Although there is no need to detail more the definition of “same” in this paper, note that it is non-trivial. As often, it is indeed crucial important to distinguish between identical and isomorphic objects: inconsistencies arise otherwise, for example if we allow the infinite right unfolding of \((\mu P. \nu Q. P)\) by mistaking repeated occurrences of \((\nu Q. \mu P. \nu Q. P)\) for successive unfoldings of the same greatest fixed point.
Notice that the variables $x$ do not occur in $\Gamma$. These rules allow to use the current elements of the context in the invariance proof, but keep it entirely separated\(^3\) from the invariant: $\Gamma$ remains constant throughout the iterative construction of $S$.

The reason for usually working only with the restricted form is that $\Gamma$ can in fact be embedded explicitly in the (co)invariant $S$ if needed. For example, the generalized induction rule can be recovered from the simple one by choosing the invariant $S' := \lambda x. \forall \Sigma. \forall \Gamma \supset Sx$ — and dually for coinduction, $\exists \Sigma. \forall \Gamma \wedge Sx$. But this expressiveness result leaves some questions open. Clairambault notes that in order to embed System $T$ in $\mu LJ$, one would have to argue how the encoding of the unrestricted induction behaves during cut-elimination, which is extremely complicated. The question we ask here is that of finding a good direct focusing treatment of the unrestricted rules: as often, an encoding does not yield a satisfying answer.

It seems impossible to treat our extended (co)induction principles as asynchronous rules. Indeed, they do not even permute with themselves — unsurprisingly, when attempting to permute the rules, we find ourselves in need of a contraction. It can, however, be treated synchronously in a $\nu$-focused system equipped with the following fixed point rules (in addition to initial rules in the synchronous phase, as well as unfoldings and freezing in the asynchronous phase):

\[
\begin{array}{c}
\frac{\Sigma; \Gamma \vdash S \vdash P}{\Sigma; \Gamma \vdash \mu b \vdash P} & \frac{\Sigma; \Gamma \vdash S \vdash P}{\Sigma; \Gamma \vdash \nu b \vdash P} & \frac{\Sigma; \Gamma \vdash S \vdash P}{\Sigma; \Gamma \vdash \nu b \vdash P}
\end{array}
\]

This observation is quite surprising. It not only confirms the validity of $\nu$-focusing, which seems incidental in some ways, but does not even admit the expected $\mu$-focusing. The synchronous treatment of generalized (co)induction rules is also surprising since it seems that choices in $\Gamma$ would not permute under (co)induction. The key here is to obtain the right dyadic presentation of those rules: only the non-linear part of the sequent should be copied in the (co)invariance premise (which is, once more, unsurprising from the linear viewpoint). Finally, as in the previous $\nu$-focusing system, balancing has to be interleaved with the elimination of contractions in the proof of completeness\(^4\).

### 5 Conclusion

We have presented and discussed the design of focused proof systems for fixed points in an intuitionistic setting. This brings out the question of the nature of focusing: we have seen that it is not directly related to (non-)invertibility, but its “deeper” explanations, e.g., in terms of games or relying linear logic’s exponentials, only account for part of the picture. Through the study of other proof systems for fixed points, we have shown that there can be a big difference between what semantic indicates and what finite proof systems allow. We hope that this paper will help to explore further these questions, as well as the extension of our observations to settings such as type theory.

---

\(^3\) This can be contrasted with the invariants used automatically by Tac: $\lambda x. \forall \Sigma. \forall \Gamma \supset x = t \supset P$, where the equality ties the induction variables $x$ to the signature $\Sigma$, sometimes too much.

\(^4\) All these observations deserve more detailed proofs: they shall be found in an upcoming extended paper.
Acknowledgments  I warmly thank Dale Miller and Alexis Saurin for many insightful discussions at various points of this work. This work has been supported in part by the National Science Foundation grant CCF-0917140. Opinions, findings, and conclusions or recommendations expressed in this papers are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References


